Multicloud Resource Allocation: Cooperation, Optimization and Sharing

THÈSE N^{O} 7483 (2017)

PRÉSENTÉE LE 10 MARS 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SYSTÈMES D'INFORMATION RÉPARTIS
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Hao ZHUANG

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. K. Aberer, directeur de thèse
Prof. Ph. Cudré-Mauroux, rapporteur
Prof. H. Pan, rapporteur
Prof. B. Ford, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

To my dear parents & family members.

Acknowledgements

At this moment, a myriad of gratitude fill my heart. I would like to take this opportunity to give my sincere thanks for those who offered the generous help during my four years of PhD studies.

First and foremost, I would like to thank my supervisor, Prof. Karl Aberer, for giving me the opportunity to do the PhD studies under his guidance. His enlightening guidance and inspiring instructions over these years have helped me greatly. He gave me the great freedom to work on the topics that I felt passionate about and I am very appreciated for that.

Secondly, I would like to express my sincere gratitude to Dr. Rameez Rahman for his valuable guidance and mentorship. He not only gave me a lot of help on searching the interesting problem and thesis writing, but, more importantly, showed me how to conduct the scientific research with critical thinking as well. Thanks for your encouragement and all discussions we made. You are my life-time comrade and I wish you all the best for the future.

Thirdly, I would like to thank Prof. Boi Faltings, Prof. Ford Bryan, Prof. Philippe Cudre-Mauroux and Prof. Pan Hui for agreeing to be my thesis committee members, and for providing valuable comments and suggestions to improve my thesis work.

I would like to thank all my colleagues in LSIR, both the alumni and the current members. My story with LSIR started with Zhixian Yan, thank for your recommendation and I wish you all the best for the future. Also, great thanks to my officemate Jean-Paul and Surender, for sharing your research experience and discussions. Thanks for Xin and Thanasios, for helping me overcome the cold-start of my PhD life. My special thanks go to our CloudSpaces team, Hamza and Rameez, thank for all technical discussions, debates and your great efforts to our project. My gratitude is extended to other members in LSIR, Tian, Michele, Julia, Jean-Eudes, Julien, Thanh, Amit, Alex, Alevtina, Martin, Remi, Panayiotis, Berker, Hung, Tri, Alexandra, Matteo, Mehdi and all members in our lab. Last but not least, many thanks to Chantal for all the help and supports throughout my stay here.

Also, I would like to thank our small Chinese community BC lunch group members, Xifan Tang, Xiaolu Sun, Tian Guo, Jian Zhang, Zhou Xue, Bin Fan, Hanjie Pan, Cheng Wang, Bin Jin, Jingjing Wang, Runwei Zhang, Xinchao Wang, Bin Ding, Wei Zhuo, Shenqi Xie, Ye Pu, Min Ye, Monkey King and Zhufei Chu. We always share the ideas as well as stories during the lunch time, which kept me high motivated and enthusiastic.

A special gratitude and love goes to my big family. I thank my parents for their deep abiding love. I thank my two elder sisters Jing Zhuang and Yan Zhuang, for their care and also my lovely niece and nephew, Shiqi and Shengbo. Thank my two brothers-in-law, Feng Mao

and Yang Yu, for their love to our family. This thesis can be impossible without all your supports.

Last but not the least, I dedicate this thesis to my wife, Huizhen LI, and my soon-to-be born baby girl Xixi. I am indebted to my wife for her constant love, encouragement, understanding and supports during the past 11 years. My dearest sweety, we have journeyed far on life's path together. I cannot wait to see what wonderful and colorful trails await us in our future. Also, our incoming baby girl Xixi, as your name suggests, we hope you can live a happy and healthy life. You will be always the source of our happiness and we love you forever and ever!

Hao Zhuang
Lausanne, 8^{th} *January, 2017*

Abstract

Nowadays our daily life is not only powered by water, electricity, gas and telephony but by "cloud" as well. Due to the high penetration of cloud-based services or applications into every aspect of our life and the unprecedented increase of digital data, cloud computing becomes the 5th utility that processes and stores these applications and data. Big cloud vendors such as Amazon, Microsoft and Google have built large-scale centralized data centers to achieve economies of scale, on-demand resource provisioning, high resource availability and elasticity. However, those massive data centers also bring about many other problems, *e.g.,* bandwidth bottlenecks, privacy, security, huge energy consumption, legal and physical vulnerabilities. One of the possible solutions for those problems is to employ multicloud architectures. In this thesis, our work provides research contributions to multicloud resource allocation from three perspectives of *cooperation*, *optimization* and *data sharing*. We address the following problems in the multicloud: how resource providers cooperate in a multicloud, how to reduce information leakage in a multicloud storage system and how to share the big data in a cost-effective way. More specifically, we make the following contributions:

Cooperation in the decentralized cloud. Recently due to increasing concerns on the privacy and data control, many small data centers (SDCs) established by different providers are emerging in an attempt to meet demand locally. However, SDCs can suffer from resource in-elasticity due to their relatively scarce resources, resulting in a loss of performance and revenue. In this work, we propose a decentralized cloud model in which a group of SDCs can cooperate with each other to improve performance. Moreover, we design a general strategy function for SDCs to evaluate the performance of cooperation based on different dimensions of resource sharing. Through extensive simulations using a realistic data center model, we show that the strategies based on *reciprocity* are more effective than other strategies, *e.g.*, those using prediction based on historical data. Our results show that the reciprocity-based strategy can thrive in a heterogeneous environment with competing strategies.

Multicloud optimization on information leakage. Many schemes have been recently advanced for storing data on multiple clouds. Distributing data over different cloud storage providers (CSPs) automatically provides users with a certain degree of information leakage control, for no single point of attack can leak all the information. However, unplanned distribution of data chunks can lead to high information disclosure even while using multiple clouds. In this work, we firstly study an important information leakage problem caused by unplanned data distribution in multicloud storage services. Then, we present *StoreSim*, an information leakage aware storage system in multicloud. StoreSim aims to store syntactically similar data on the same cloud, thereby minimizing the user's information leakage across multiple clouds. We design an approximate algorithm to efficiently generate similarity-preserving signatures for data chunks based on MinHash and Bloom filter, and also design a

function to compute the information leakage based on these signatures. Next, we present an effective storage plan generation algorithm based on clustering for distributing data chunks with minimal information leakage across multiple clouds. Finally, we evaluate our scheme using two real datasets from *Wikipedia* and *GitHub*. We show that our scheme can reduce the information leakage by up to 60% compared to unplanned placement. Furthermore, our analysis in terms of *system attackability* demonstrates that our scheme makes attacks on information much more complex.

Smart data sharing. Moving large amounts of distributed data into the cloud or from one cloud to another can incur high costs in both time and bandwidth. The optimization on data sharing in the multicloud can be conducted from two different angles:

- **Inter-cloud scheduling.** Existing centralized solutions for data sharing such as Dropbox replicating data to all interested parties is prohibitively costly, given the large size of datasets. A more practical solution is to use a Peer-to-Peer (P2P) approach to replicate data in a self-organized manner. However, existing P2P approaches focus on minimizing downloading time without taking into account the bandwidth cost. In this work, we present *CoShare*, a P2P inspired decentralized cost effective sharing system for data replication. CoShare allows users to specify their requirements on data sharing tasks and maps these requirements into resource requirements for data transfer. Through extensive simulations, we demonstrate that CoShare finds the desirable tradeoffs for a given cost and performance while varying user requirements and request arrival rates.
- **Making big data smaller.** The sheer size of big data imposes great challenges on storing, sharing and processing such data in the multicloud. These challenges can be addressed by data summarization which transforms the original dataset into a smaller, yet still useful subset. In this work, we take Twitter data and its applications based on topic models as a case study. We aim to reduce the size of the Twitter dataset while preserving topics in the original big dataset. Existing work finds such small subsets with objective functions based on data properties such as representativeness or informativeness but does not exploit social contexts, which are distinct characteristics of social data. Through analyzing Twitter data, we discover two social contexts which are important for topic generation and dissemination, namely (i) *CrowdExp* topic score that captures the influence of both the crowd and the expert users in Twitter and (ii) *Retweet* topic score that captures the influence of Twitter users' actions. We conduct extensive experiments on two real-world Twitter datasets using two applications. The experimental results show that, by leveraging social contexts, our proposed solution can reduce the total size of data without the performance degradation on topic-related applications.

Keywords: *Mutlicloud resource allocation, decentralized cloud, multicloud storage system, multicloud optimization, data sharing, small data center, data transfer, information leakage, system attackability, data summarization, social context*

Résumé

De nos jours, notre vie quotidienne est non seulement alimentée par l'eau, l'électricité, le gaz et la téléphonie mais également par "l'informatique en nuage". En raison de la forte pénétration des services ou applications basés sur le cloud dans tous les aspects de notre vie, et de l'augmentation sans précédent des données numériques, le cloud computing devient le cinquième service qui traite et stocke ces applications et données. Les grands fournisseurs de cloud tels que Amazon, Microsoft et Google construisent des centres de données centralisés pour réaliser des économies d'échelle, et fournir des ressources à la demande en abondance et avec une grande élasticité. Cependant, ces centres de données massifs engendrent également de nombreux problèmes : embouteillage de la bande passante, protection de la vie privée, sécurité des données, une consommation énorme d'énergie, vulnérabilité juridique et physique. Une des solutions possibles pour ces problèmes est d'utiliser des architectures multi-cloud. Dans cette thèse, nous proposons l'allocation de ressources multi-cloud selon trois perspectives : *coopération*, *optimisation* et *partage de données*. Nous abordons ainsi les problèmes suivants : (1) comment les fournisseurs de ressources collaborent dans un multi-cloud; (2) comment réduire les fuites d'information dans un système de stockage multi-cloud; (3) comment partager un grand volume de données de façon rentable. Plus précisément, nous apportons les contributions suivantes :

Coopération dans le nuage décentralisé. Récemment, en raison des inquiétudes grandissantes concernant la vie privée et le contrôle des données, de nombreux petits centres de données (SDC) établis par différents fournisseurs émergent pour tenter de répondre à la demande locale. Cependant, les SDC peuvent souffrir de l'élasticité des ressources en raison de leurs ressources relativement limitées, ce qui entraîne une perte de performance et de revenus. Dans ce travail, nous proposons un modèle de nuage décentralisé dans lequel un groupe de SDC peut coopérer pour améliorer les performances. De plus, nous introduisons une fonction de stratégie générale pour les SDC afin d'évaluer la performance de la coopération basée sur les différentes dimensions du partage des ressources. Grâce à de vastes simulations utilisant un modèle de centre de données réaliste, nous montrons que les stratégies basées sur la *réciprocité* sont plus efficaces que d'autres stratégies, comme par exemple celles utilisant la prédiction basée sur des données historiques. Nos résultats montrent que la stratégie basée sur la réciprocité peut prospérer dans un environnement hétérogène avec des stratégies concurrentes.

Optimisation multi-cloud sur les fuites d'information. De nombreux systèmes ont récemment été développés pour stocker des données sur plusieurs nuages. Cette distribution de données sur différents fournisseurs de stockage en nuage fournit automatiquement aux utilisateurs un certain degré de contrôle des fuites d'informations, car aucun point d'attaque unique ne

peut divulguer toutes les informations. Toutefois, la distribution non planifiée des blocs de données peut engendrer une divulgation d'informations élevée, même en utilisant plusieurs nuages. Dans ce travail, nous étudions d'abord un important problème de fuite d'information causé par la distribution non planifiée des données dans les services de stockage multi-cloud. Ensuite, nous présentons *StoreSim*, un système de stockage de détection de fuites d'informations en multi-cloud. StoreSim vise à stocker des données syntaxiquement similaires sur le même nuage, ce qui minimise les fuites d'informations de l'utilisateur sur plusieurs nuages. Nous introduisons un algorithme d'approximation pour générer efficacement des signatures de préservation de similarité pour les blocs de données basés sur le filtre MinHash et Bloom, accompagné d'une fonction pour calculer la fuite d'information basée sur ces signatures. Ensuite, nous présentons un algorithme efficace de génération de plans de stockage, basé sur le clustering, pour la distribution de blocs de données en minimisant les fuites d'informations sur plusieurs nuages. Enfin, nous évaluons notre système en utilisant deux ensembles de données réels de *Wikipedia* et *GitHub*. Nous montrons que celui-ci peut réduire les fuites d'information jusqu'à 60% par rapport à un placement non planifié. De plus, notre analyse en termes d'*attaques du système* démontre que notre système rend les attaques sur les informations beaucoup plus complexes.

Un partage de données intelligent. Le déplacement de grandes quantités de données distribuées vers le nuage ou d'un nuage vers un autre peut entraîner des coûts élevés en termes de temps et de bande passante. L'optimisation du partage des données dans le multi-cloud peut être réalisée sous deux angles différents:

- **Une organisation inter-cloud.** Les solutions existantes et centralisées pour le partage de données telles que Dropbox qui duplique les données pour toutes les parties intéressées sont prohibitivement coûteuses, étant donné la grande taille des ensembles de données. Une solution plus pratique consiste à utiliser une approche pair-à-pair (P2P) pour reproduire les données de manière auto-organisée. Toutefois, les approches P2P existantes se concentrent sur la minimisation du temps de téléchargement sans tenir compte du coût de bande passante. Dans ce travail, nous présentons *CoShare*, un P2P qui est inspiré par un système de partage décentralisé pour la réplication des données. CoShare permet aux utilisateurs de spécifier leurs besoins sur les tâches de partage de données et de cartographier ces exigences en ressources requises pour le transfert de données. Grâce à de nombreuses simulations, nous montrons que CoShare peut trouver les compromis souhaitables par rapport à un coût et une performance donnés, tout en variant les exigences des utilisateurs et les taux d'arrivée des demandes.
- **Rendre les données volumineuses plus petites.** Le grand volume de données impose des défis considérables pour le stockage, le partage, et le traitement de ces données dans le multi-cloud. Ces défis peuvent être résolus en réalisant une synthèse des données, transformant ainsi l'ensemble des données original en un sous-ensemble plus petit, mais encore utile. Dans ce travail, nous considérons les données de Twitter et ses applications basées sur les modèles thématiques (*topic models*) comme étude de cas. L'objectif est de réduire la taille du jeu de données de Twitter tout en préservant

les thèmes issus du jeu de données original. Les travaux existants définissent ces petits sous-ensembles avec des fonctions-objectifs basées sur des propriétés telles que la représentativité ou l'informativité des données, mais n'utilisent pas le contexte social qui met pourtant en évidence des caractéristiques distinctes dans ces données. En analysant les données de Twitter, nous avons découvert deux contextes sociaux d'importance pour la génération et la dissémination de thèmes, à savoir : (1) le score thématique *CrowdExp* qui capture l'influence de la foule et des utilisateurs experts dans Twitter; (2) le score thématique *Retweet* qui capture l'influence des actions des utilisateurs de Twitter. Nous avons ensuite conduit des expériences approfondies sur deux ensembles de données réels de Twitter en utilisant deux applications. Les résultats expérimentaux montrent que, en tirant profit des contextes sociaux, la solution proposée permet de réduire la taille totale des données sans dégrader les performances sur ces applications basées sur des modèles thématiques.

Mots-clés: *allocation des ressources multi-cloud, nuage décentralisé, système de stockage multi-cloud, optimisation multi-cloud, partage de données, petit centre de données, transfert de données, fuite d'information, attaquabilité du système, résumé des données, contexte social*

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Chapter 1

Introduction

It's stupidity. It's worse than stupidity: it's a marketing hype campaign. Somebody is saying this is inevitable — and whenever you hear somebody saying that, it's very likely to be a set of businesses campaigning to make it true.

Richard Stallman thinks Cloud Computing is a trap for users, quoted in The Guardian, September 29, 2008

1.1 Background

Nowadays our daily life is not only powered by water, electricity, gas and telephony but by "cloud" as well. Due to the high penetration of cloud-based services or applications into every aspect of our life and the unprecedented increase of digital data, cloud computing becomes the 5th utility that processes and stores these applications and data. Big vendors such as Amazon, Microsoft and Google have already built large-scale data centers to deliver compute infrastructure as a service. The computing service gains strength in the IT marketplace quickly due to its on-demand resource provisioning, high availability and elasticity. These features allow cloud users such as cloud service providers to access resources (*i.e.*, compute, storage and network) in a pay-as-you-go manner and to meet varying demands without upfront resource commitments.

Though many advantages over previous computing paradigm, cloud computing is not without its problems. For example, in order to accommodate load spikes and to maintain high availability, public cloud providers have to provisioning more resources than necessary, leading to the low resource utilization. There are also many other problems such as data privacy concerns, huge energy consumption, bandwidth bottlenecks, legal issues, physical vulnerabilities, to name but a few. In the following, we will further discuss three key issues in this thesis.

The need of decentralization. Currently, centralized cloud computing paradigm is dominant in cloud markets, given its ability to aggregate large computing and storage infrastructures to obtain economies of scale. For many companies, especially small and medium enterprises (SMEs), they can achieve high

cost savings in their service deployment and maintenance as well as high service availability and scalability through renting computing resources from centralized cloud providers. However, this centralized architecture is criticized for grabbing the control of user's services and data. Adoption of the centralized cloud means a compromise on application control and data privacy. In addition, from the perspective of cloud providers, they also suffer many issues with centralization such as huge energy consumption, large amount of data transportation and data security problems. Thus, in recent years, there is an increasing demand on a decentralized cloud approach to solving these concerns that are caused by centralization.

Data security and privacy concerns. Once adopting cloud services, users are forced to face utility & privacy dilemma. On the one hand, cloud users can reduce their costs through renting resources from the cloud. On the other hand, cloud users might highly depend on cloud providers, if their applications and data are managed in the cloud. Furthermore, many evidences show that centralized cloud services exploit users' personal data for targeting advertisements or business analytics on market research[1]. Existing work on protecting user's cloud data includes encrypting data in both server side and client side. However, while encryption-based approach eliminates most of the data privacy concerns, it also prevents users from benefiting existing cloud services [2]. For example, if all the documents are encrypted in the cloud, it is difficult to share these encrypted documents for team collaboration. Other approaches to improving data privacy is to employ multicloud storage in which no single cloud has all user's data [3]. These protection techniques force attackers to increase their costs and finally to give up.

Data sharing bottleneck. We are living in the era of big data. Cloud-based applications are becoming more and more data intensive. The data of these applications are generated all the time by users or their associated devices and sensors which are widely distributed in different locations. It is a non-trivial task to move these massive distributed data efficiently from the end to the centralized cloud. In addition, the high cost on the network bandwidth further complicates data sharing problems. It is reported that, to overcome data transfer bottleneck in network bandwidth and its associated high cost, Netflix ships its data to the Amazon cloud by overnight delivery services [4]. Therefore, it gives us opportunities to optimize the data transfer over the data center networks. For example, we can minimize the transfer time or bandwidth cost for data sharing tasks. In the real scenario, we may further manage the tradeoffs between the transfer time and the transfer cost. Besides optimizing data transfer over the network, an early optimization before data sharing is to reduce the total size of data that needs to be transferred. For example, most of cloud storage providers employ data deduplication to avoid retransmission of the duplicate data, thereby reducing the total network transfer.

One possible solution to tackle above three problems is to leverage multicloud architecture. Multicloud is a cloud model that allows on-demand access to resources aggregated from different cloud providers. It is a fact that the performance and prices of different cloud providers vary significantly [5]. There is no single cloud provider that can satisfy all users' requirements for service quality with each provider's SLA (Service Level Agreement). Multicloud model allows cloud users to make better use of each cloud's advantages whilst minimizing the weakness of each cloud, thereby improving service performance. Many related work investigates multicloud from different dimensions. As is shown in Figure 1.1, we summarize their work into four different types:

- *Geographically distributed cloud.* Nowadays, big cloud vendors such as Amazon, Google and Microsoft provide their multicloud solution with building massive data centers in geographically distributed locations world widely. This multicloud can satisfy various business requirements of a

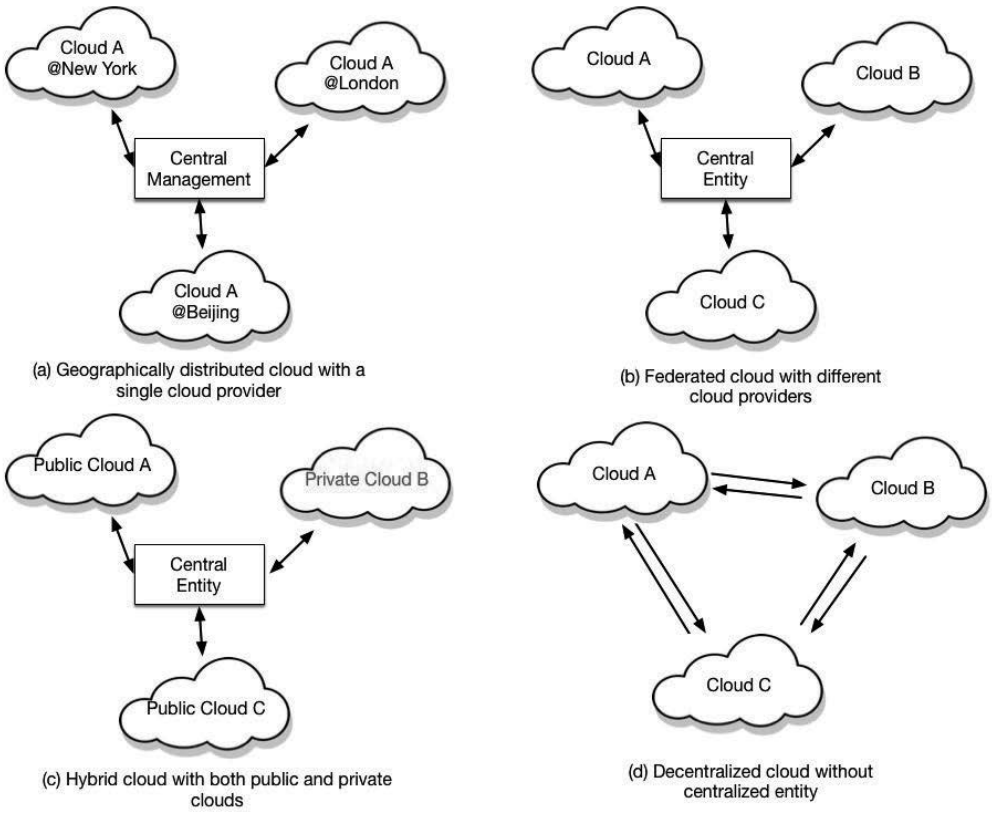

Figure 1.1: Different types of multicloud

wide range of cloud services, especially location-based services. For example, service providers can serve their customers from multiple geographic regions, thereby minimizing service response latency and improving service availability. Nevertheless, all distributed clouds in this multicloud are belonged to the same cloud vendor, as shown in Figure 1.1 (a). The cloud users still suffer the vendor lock-in problem, for the reason that their services and data are centrally managed by the same vendor.

- *Federated cloud.* This multicloud model aggregates resources from different cloud providers into a single resource pool with a uniform access, as shown in Figure 1.1 (b). In other words, cloud federation is a virtual *cloud of clouds*, which adopts a combination of diverse clouds to overcome vendor lock-in problems. Thus, it is vital important to enable cloud interoperability among different cloud providers to manage heterogeneous resources. One simple solution is to implement a centralized cloud broker which manages a set of public clouds' APIs, such as Apache jclouds [6]. An alternative solution is to implement a common interoperability layer that allows different cloud to negotiate resource sharing [7].

- *Hybrid cloud.* There are many concerns that prevent the adoption of the cloud such as compliance

issues, performance requirements, and security restrictions. The need for local in-house data centers is a fact of life. Hence, for those companies which have private data centers, they prefer to employ a hybrid cloud as their multicloud solution. As it is shown in Figure 1.1 (c), hybrid cloud is composed of at least one private and one public cloud from different cloud providers. Consequently, hybrid cloud users have to consider what services and data have to be kept locally and what can be processed remotely. With the help of hybrid cloud, enterprises can retain control of their own private data centers while sending non-mission-critical workloads to the public clouds.

- *Decentralized cloud.* A decentralized cloud consists of heterogeneous resources from both data centers and individuals. The resources are managed in a peer-to-peer manner and there is no central entity as shown in Figure 1.1 (d). The similar terms such as Peer-to-Peer (P2P) Cloud, volunteer cloud, edge cloud are essentially to describe how to manage heterogeneous resources in a decentralized manner based on P2P and cloud computing. The decentralized cloud is also a highly virtualized platform that provides compute, storage, and network services among different peers. The power of virtualization makes it possible to manage resources in a virtual execution environment, which facilitates resource trading and sharing. All the peers make their decisions on resource allocation in a strategic setting where various strategies based on different preferences (e.g., location proximity, privacy concerns, legal constraints) are employed by peers. The strategy which is used by a participant may depend on what the others are using and vice versa. At present, there are several decentralized clouds, among which Symform [8] and Storj [9] are mainly for storage and SpotCloud [10] is for both compute and storage resource trading.

The above four type of multicloud models together with traditional centralized cloud model complement each other to satisfy various requirements from cloud users. In this thesis, we will focus on investigating how to exploit different types of multicloud to solve key issues in the centralized cloud. We firstly introduce the challenges of our work in the next section.

1.2 Mutlicloud Resource Allocation and the Challenges

Within the multicloud, we need to manage heterogeneous resources (*i.e.*, computing, storage and bandwidth) from different cloud providers in a cost-effective manner. In this thesis, we study multicloud resource allocation in three different settings: (i) compute resource sharing in the decentralized cloud (chapter 3); (ii) reducing information leakage in the multicloud storage system (chapter 4) and (iii) data sharing in the decentralized cloud (chapter 5 and 6). As it is shown in Figure 1.2, we investigate compute sharing and data sharing in the decentralized cloud while optimizing information leakage in a hybrid cloud with a centralized optimizer. Each setting raises different research questions and we will further discuss these questions and its challenges in the following.

(i) Compute resource sharing in the decentralized cloud. In this study, we investigate a decentralized multicloud model to avoid severe issues (e.g., legal, privacy, and data control) that can arise with the adoption of massive centralized data centers. In particular, we primarily target small data centers (SDCs) with dozens or hundreds of server machines. For example, in a country like Switzerland with various cantons, one could imagine the emergence of multiple small data centers in various companies and institutions to obviate the reliance on big enterprise data centers (e.g., Amazon EC2, Microsoft Azure, etc.)

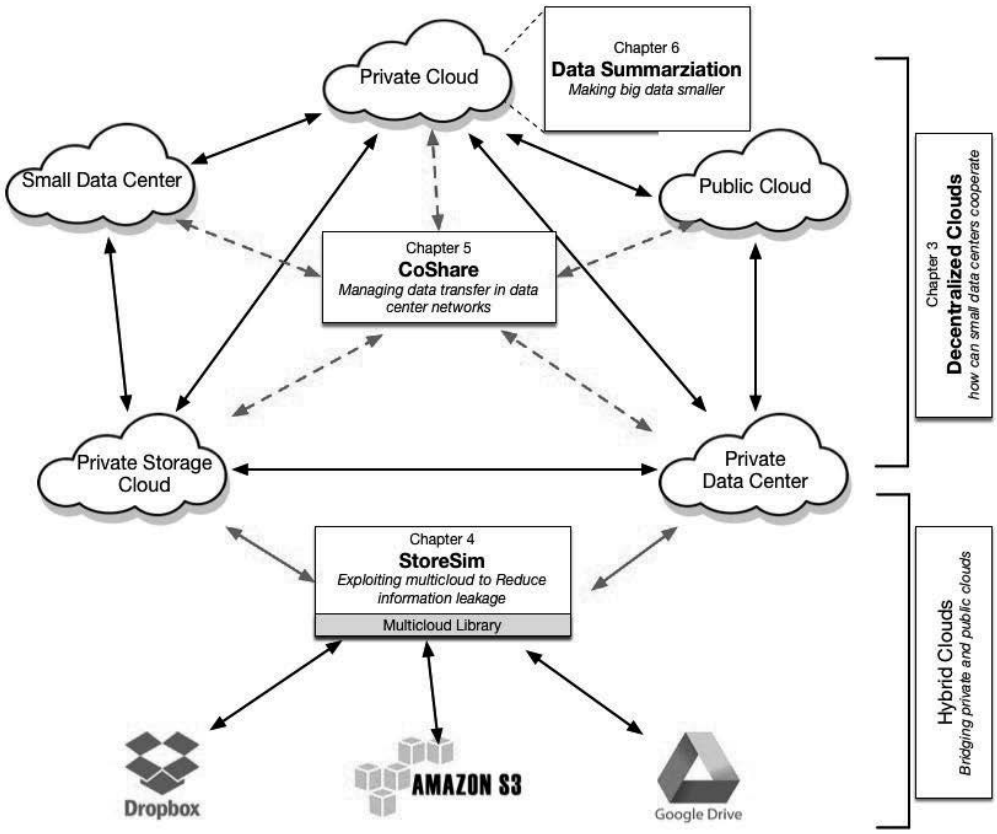

Figure 1.2: Overview of the work and structure of this thesis

and to maintain control over local data. Nevertheless, they also confront the same problems of resource over-provisioning, low resource utilization and high maintenance cost. Moreover, due to the limited capacities, those small-scale data centers alone may not be able to provide elastic services with high availability during periods of peak demands. In some cases, they could even act as resource consumers to seek resources from other data centers. Therefore, this motivates small data centers to collaboratively provide cloud services to improve the efficiency of resource allocation. In this part, we envision such a decentralized cloud with a swarm of networked small data centers which cooperate with each other under varying workloads and which also employ various strategies of resource allocation. Specifically, we need to solve the following research questions and their challenges:

- *What are the incentives of SDCs for cooperation within the decentralized cloud?* In the decentralized cloud, each SDC makes their decisions on resource allocation independently, without any centralized entity. The first immediate question is to study the incentives of different customers on why they want to contribute their resources and to utilize others' resources. Existing work have focused on cost minimization or revenue maximization for incentivizing data centers to cooperate

[8]. However, the incentives why cloud providers develop cooperation with each other are much more diverse due to the high heterogeneity in the decentralized cloud. In a dynamic setting of data centers with varying workload patterns and changing sociopolitical and legal realities, prices and revenue maximization are not the sole (or at times even relevant) criterion that can determine cooperation. Factors such as location proximity, workload similarity, privacy concerns etc., can be equally, if not more, important considerations for resource exchange and allocation between small data centers. Thus, it is essential for us to find out these incentives so that we can better improve the efficiency of resource allocation.

- *how is the performance of cooperation affected due to different strategies adopted by SDCs?* Since SDCs are of high heterogeneity with different workload patterns, resource capacity and geographic location, their cooperation strategies in the decentralized cloud may vary greatly. For example, some SDCs may consider the history of past collaboration while others may attach more importance to the prediction of future workload. It is challenging to evaluate the performance of different strategies, since it involves a wide range of factors such as computing capacity, network latency and workload patterns. In addition, social factors such as reciprocity, altruism and friendship further complicate this problem. Thus, it is a non-trivial task to design an approach to evaluate the performance of cooperation with incorporating these diverse factors.

- *Is there any strategy that can thrive in a heterogeneous environment?* Given many possible cooperation strategies, an immediate question is whether there is a best cooperation strategy. In the real situation, which strategy is best depends partly on what strategies the other SDCs are using. In the same way, the strategies used by others may well depend on what strategy you are choosing. For example, each SDC in the decentralized cloud will keep change its strategy if they observe the poor performance of current strategy. In this way, they may find out the best strategy that can thrive in a heterogeneous environment. There is a competition among different cooperation strategies adopted by different SDCs. Hence, we need to design an adaptive strategy selection algorithm and help SDCs discover the fittest strategy with the best cooperation performance in the decentralized cloud.

(ii) Reducing information leakage in the multicloud storage system. Cloud-based storage and file sharing services have become near-ubiquitous among both personal and enterprise users. Service providers such as Dropbox, Google Drive and Amazon S3, have gained popularity due to their easy-to-use interface and low storage cost. However, these centralized cloud storage services are also criticized for grabbing the control of users' data, which allows them to run analytics for marketing and advertising [1]. In addition, the information in user's data can be leaked e.g., by means of malicious insiders, backdoors [2], bribes and coercion [3]. One possible solution to reduce the risk of information leakage is to employ multicloud storage systems [4, 5, 6, 7] in which no single point of attack can leak all the information. In this study, we consider a multicloud storage system using storage resources from a federation of multiple public clouds. The optimization on reducing information leakage is conducted by a centralized broker, which also manages cloud interoperability among public clouds. To realize this multicloud storage system, we need to tackle the following two challenges:

- *how to efficiently measure information leakage?* Information is an abstract concept which needs to be further defined given a specific context and data. In this study, we need to measure the information leakage in the process of storing user's data in the multicloud. There are two main methods to measure the information leakage. One is based on information theory which is to measure the information gains based on statistical divergences such as Kullback–Leibler divergence. The other approach is based on similarity that information leakage is measured as the dissimilar information such as Jaccard similarity. However, these methods are compute intensive which cannot be directly applied in the multicloud storage system. For example, if we measure information leakage in terms of Jaccard similarity, we need to represent user's data as a set of words. The set operations for measuring pairwise information leakage based on Jaccard similarity can be quite expensive. In addition, with the significant increase in the number of data stored in the multicloud, the number of pairs also increases quadratically. Thus, we need an efficient algorithm to compute the information leakage with less computation overhead.

- *how to reduce information leakage in a multicloud storage system?* After we determine how to measure information leakage, the next step is to generate a storage plan which defines how to distribute user's data across multiple clouds. The storage plan can be generated in terms of users' preference and quality of service (QoS) factors. For example, a simple storage plan is based on round robin which distributes users' data to each cloud in equal portions. Apparently, round robin based storage plan has a good balance on storage load but does not take information leakage into account. In this part, we evaluate the goodness of storage plan with respect to the information leakage. We aim to search a storage plan with the minimal information leakage to each cloud in the multicloud. This search problem is challenging due to the large search space. In addition, the search space increases with more data stored in the multicloud. Therefore, we need to design an efficient algorithm to find the optimal or near-optimal storage plan with minimal information leakage in a fast and efficient way.

(iii) Smart data sharing in the decentralized cloud. In the era of Big data, heterogeneous data are generated at a huge scale from many different domains, such as online social networks (OSNs), sensor networks, medical images, scientific measurements (e.g., CERN [1]) and Internet-wide measurements. This brings new challenges on how to efficiently share these big data for research community and industry. First, existing data sharing architectures are not suitable for big data sharing. The centralized architecture of data sharing such as Dropbox requires huge upfront investment on dedicated server machines and high bandwidth whilst existing P2P approaches focus on maximizing network throughput without taking the bandwidth cost into account. In addition, the sheer size of data further complicates the process of data sharing in the multicloud. Hence, we need to design a cost-effective data sharing system to replicate data in a smart way. In order to achieve this, we optimize data sharing from two perspectives. First, we formulate a linear programming problem to optimize data sharing among the data center networks. Second, within each cloud, we propose a data summarization approach to reduce the size of dataset, thereby reducing total network transfer. Specifically, we need to deal with the following two questions.

- *how to manage cost-performance tradeoffs in a data sharing system?* Data sharing among different data centers incurs a large amount of data traffic as well as a huge cost on network bandwidth.

The efficiency of a data sharing network can be measured in terms of several metrics such as bandwidth cost, network throughput, link utilization, network congestion, etc. Current data sharing systems focus on either minimizing the bandwidth cost or maximizing the network throughput. In the real settings, there usually exists conflicts in these objectives. For example, minimizing the completion time of data sharing means more cost on bandwidth. Hence, we need to design a cost-effective data sharing system that can manage the tradeoffs in the data sharing network. Current client-server approaches, such as Content Distribution Networks (CDNs) [3] for data sharing, require dedicated servers that have high storage capacity and bandwidth to maintain a centralized repository whilst P2P-based approaches fail to take bandwidth cost into account. What is needed is an approach that allows individual data centers to collaborate in a lightweight manner while providing cost and performance-effective data transfers, i.e., the system should find the most cost-effective solution given the performance requirements. In addition, price heterogeneity in different locations is also a crucial factor that influences the bandwidth cost. For example, the price of bandwidth in Sao Paulo is nearly two times higher than that in US/Europe area in Amazon data center. We need to design our system that can be heterogeneity awareness on the price. Finally, such system shall provide a usable interface that allows users to specify their performance goals for data sharing tasks. In other words, data sharing system can support to translate user's high-level performance goals into specific resource requirements for data sharing process.

- *Making big data smaller.* Besides centralized optimization on data transfer cost or network throughput, we can also apply some techniques to reduce the total amount of data that needs to be transferred. For example, we can apply data compression, which reduces the cost of data transfer by compressing big dataset into smaller size, or data deduplication, which avoids the same data being unnecessarily sent across the network again thereby reducing traffic load. Different from these traditional methods, we propose a data summarization framework that can summarize the original big dataset with a smaller summarized dataset. In this part, we focus on the social data and further take Twitter data as a case study. There are two objectives of data summarization task :1) the size of data is reduced and 2) the selected subset shall be useful. This task is challenging for two reasons. First, the usefulness of data depends on the different types of applications. The summarized subset obtained from data summarization should maintain (or even improve) the performance of applications. In other words, the performance of a trending topic application should not be much worse when it works on a summarized dataset as compared to its performance on the original dataset. Second, data summarization algorithm shall be time efficient. If the time cost of data summarization is less than that of transferring the original dataset, the data summarization can achieve cost-savings in both data sharing completion time and bandwidth cost.

1.3 Contributions

In addressing above research questions, this thesis makes the following contributions:

(i) Compute resource sharing in the decentralized cloud – Chapter 3: in this chapter, we propose a decentralized multicloud model with a swarm of networked SDCs which cooperate with each other under varying workloads, and which employ various strategies in order to cooperate with others. Hence, our work focuses on studying the incentives of different SDCs and on investigating the performance

of different cooperation strategies. Specifically, the main contributions of this work are summarized as follows:

- We propose a model for decentralized cloud with a swarm of networked SDCs. Three workload arrival models are introduced to simulate different levels of workload burstiness in the SDCs.
- A general strategy function that can be employed by SDCs is proposed to evaluate the performance of cooperation between the SDCs. Moreover, we design four specific strategy functions based on different dimensions: *capacity, history, prediction* and *reciprocity*, to model various conditions under which SDCs can choose to cooperate with each other.
- Through extensive simulations of realistic models of data centers, we analyze the performance of various strategies from the perspectives of both the decentralized cloud as a whole, and individual SDCs. We discover that the strategy with the best performance is the simplest strategy which is based on reciprocity. Furthermore, we design a new adaptive approach for SDCs to learn how to select a strategy effectively in a heterogeneous environment. The results reveal that most of the SDCs will eventually converge on the same strategy in order to achieve the state of stable mutual cooperation.

(ii) Reducing information leakage in the multicloud storage system – Chapter 4: in this work, we design a hybrid multicloud storage system with employing both private and public clouds. The system aims to reduce information leakage to each individual CSP and to provide mechanisms for distributing users data over multiple CSPs in a leakage aware manner. The contributions of this work can be summarized as follows:

- We firstly present *StoreSim*, an information leakage aware multicloud storage system which incorporates three important distributed entities and we also formulate information leakage optimization problem in multicloud.
- We propose an approximate algorithm, *BFSMinHash*, based on Minhash and Bloom filter to generate similarity-preserving signatures for data chunks. We also design a pairwise information leakage function based on Jaccard similarity.
- Based on the information leakage measured by BFSMinHash, we develop an efficient storage plan generation algorithm, *SPClustering*, for distributing users data to different clouds.
- Finally, we use two datasets crawled from *Wikipedia* and *GitHub*, containing files with multiple revisions, to evaluate our framework. Through extensive experiments, we show the effectiveness and efficiency of our proposed scheme for reducing information leakage across multiple clouds. Furthermore, our analysis on the system attackability demonstrates that StoreSim makes attacks on information much more complex.

(iii) Smart data sharing in the decentralized cloud – Chapter 5: in this part, our work focuses on two different aspects that enable smart data sharing in the decentralized cloud. One the one hand, we design a cost-effective data sharing system with a centralized optimizer to efficiently manage the cost-performance tradeoff of data sharing in the data center networks. The data sharing combines the advantages of both centralized and peer-to-peer data sharing. In details, we make the following contributions:

- We analyze the tradeoff curve between time and cost in a data sharing network based on different sharing plans and present a system interface that allows users to specify their requirements.
- We formulate the problem of finding a sharing plan with both cost and performance constraints and present our algorithm to find a sharing plan that matches user's preferences.
- Through extensive experiments based on our simulator, we demonstrate that CoShare is able to find the desirable tradeoffs between cost and performance given varying user requirements and request arrival rates.

Chapter 6: On the other hand, it is important to reduce the amount of data before they are transferred over the network. Unlike traditional data compression or data deduplication techniques, we design a data summarization model for social data that can make a smaller summary for the big dataset, thereby reducing the total data transfer. In this part, we take Twitter data as a case study and focus on the applications based on topic model. Thus, we need to summarize big Twitter data with a small subset which can preserve topics in the original dataset. We also explore how to exploit social contexts for data summarization task and the contributions are described as follows:

- Through analyzing statistical topic models, we present an objective function for preserving topics in Twitter data. We also present two challenges related to parameter estimation and the size of the search space for this function. To solve these two challenges, we first design a submodular model, called *E-model*, based on information entropy. Apart from solving the parameter estimation challenge, *E-model* also provides a lower bound for searching the optimal subset with a greedy algorithm.
- Based on the *E-model*, we further devise our novel model, *S-model*, which incorporates two important social contexts that influence topic generation and dissemination, namely *CrowdExp* and *Retweet* topic score. The former considers the influence of both the experts and the crowd (the majority of the users) while the latter captures the influence of Twitter users' actions (i.e., retweet in our case).
- We conduct experiments on real-world Twitter datasets using two different applications, Topic Discovery and Tweet Classification. The experimental results demonstrate that, by leveraging social contexts, *S-model* can help topic-preserving data summarization and further improve application performance by up to 18%.

1.4 Thesis Organization

As shown in the Figure 1.2, the remainder of the thesis is organized as follows: Chapter 2 presents a survey of literature related to research challenges addressed in this thesis work. In the chapter 3, we propose a decentralized cloud in which a group of networked SDCs can work collaboratively. We focus on studying cooperation incentives and on evaluating the performance of different cooperation strategies for sharing computing resources in the decentralized cloud. In Chapter 4, we present *StoreSim* which reduces information leakage in the multicloud storage system. Specifically, we propose an approximate algorithm to measure the information leakage based on Jaccard distance and design an efficient storage

plan generation algorithm for distributing users' data to different public clouds. Chapter 5 and 6 focus on smart data sharing in data center networks. Chapter 5 introduce a peer-to-peer data sharing system, named *CoShare*, with a centralized optimizer on allocating network bandwidth in a decentralized cloud. Chapter 6 proposes data summarization model with considering social contexts, as an approach to reduce the total amount of data that need to be transferred over the network. Finally, chapter 7 concludes our thesis and discusses the future work.

1.5 Selected Publications

- H. Zhuang, R. Rahman, H. Pan and K. Aberer. "Data Summarization with Social Contexts". *25th ACM International Conference on Information and Knowledge Management* (CIKM 2016)
- H. Zhuang, R. Rahman, H. Pan and K. Aberer. "StoreSim: Optimizing Information Leakage in Multicloud Storage Services". *7th IEEE International Conference on Cloud Computing Technology and Science*, Vancouver, BC, Canada, 2015. (**Best Student Paper Award**)
- T. Guo, J.-P. Calbimonte, H. Zhuang and K. Aberer. "SigCO: Mining Significant Correlations via a Distributed Real-time Computation Engine". *IEEE International Conference on Big Data*, SANTA CLARA, 2015.
- H. Zhuang, I. Filali, R. Rahman and K. Aberer. "CoShare: A Cost-effective Data Sharing System for Data Center Networks". *2015 IEEE International Conference on Collaboration and Internet Computing*, Hangzhou, China, 2015.
- H. Zhuang, R. Rahman and K. Aberer. "Decentralizing the Cloud: How Can Small Data Centers Cooperate?" *14th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2014.
- T. Guo, T. G. Papaioannou, H. Zhuang and K. Aberer. "Online indexing and distributed querying model-view sensor data in the cloud". *The 19th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2014.
- Z. Ou, H. Zhuang, A. Lukyanenko, J. K. Nurminen and P. Hui. "Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds", in *IEEE Transactions on Cloud Computing*, 2013.
- H. Zhuang, X. Liu, Z. Ou and K. Aberer. "Impact of Instance Seeking Strategies on Resource Allocation in Cloud Data Centers". *Sixth International Conference on Cloud Computing (CLOUD '13)*, San Jose, USA, 2013.

Chapter 2

State of the Art

If a man keeps cherishing his old knowledge,so as continually to be acquiring new, he may be a teacher of others.

Confucius

In this chapter, we present the state-of-the-art work related to the thesis in detail. Firstly, we will introduce the multicloud from the point of architecture view and then further discuss how different multicloud complement each other in section 2.1. Then we will focus on discussing three interactions in the multicloud, namely cooperation (in section 2.2), optimization (in section 2.3) and sharing (in section 2.4). For each topic, we will discuss characteristics, issues and state-of-the-art approaches. In addition, we also introduce a list of related projects in the multicloud. Finally, we conclude this chapter by differentiating our work from existing work in section 2.6.

2.1 Multicloud Resource Allocation

The traditional centralized cloud brings a lot of issues which can be solved by multicloud solutions. In this section, we firstly discuss the multicloud resource allocation from both centralized and decentralized perspectives. Then we will discuss four different types of multicloud and illustrate how these multicloud architectures complement to the existing cloud computing architecture. Finally, we will outline three representative interactions among different entities in the multicloud resource allocation.

2.1.1 Centralized or Decentralized

From the architecture point of view, multicloud can be generally classified into two categories: centralized and decentralized. On the one hand, the resources in a centralized infrastructure are managed by a centralized entity, *e.g.*, a company or an organization. That means cloud providers in the centralized architecture are usually big vendors with rich assets and technologies, which allows them to build up massive data centers. On the other hand, the resources in a decentralized infrastructure are managed by different entities. Each entity in the decentralized architecture is free to negotiate and determine the way of resource sharing. In other words, decentralized computing infrastructure democratizes the supply side

of cloud computing by allowing small and medium-sized enterprises (SMEs) and other new participants to become resource providers.

The cloud computing begins with centralized architectures, which can create effective economies of scale [4]. The representative cloud vendors are Amazon EC2, Google compute engine and Microsoft Azure. However, with continually evolving requirements of various cloud services, one single cloud cannot meet all these requirements [5]. For example, a user in New York will find that it takes too much time to load a service which is hosted in California. To meet this demand, these big vendors also offer their cloud services in several geographically distributed data centers to meet users' diverse demands, *e.g.*, using location proximity to reduce response latency. However, this does not change the inherent nature of centralized architecture. Those issues raised by centralized architecture are not solved, such as huge energy consumption, bandwidth bottlenecks, privacy, security, legal and physical vulnerabilities.

To solve these problem, we need a decentralized computing infrastructure. We envision a decentralized cloud in which the resources are contributed and shared by a group of small data centers (SDCs). The decentralized cloud is also a highly virtualized platform that provides compute, storage, and network services to users. The power of virtualization makes it possible to manage resources in a virtual execution environment, which facilitates resource trading and sharing. All SDCs in the decentralized cloud make their decisions on resource allocation in a strategic setting where various strategies based on different preferences (e.g., location proximity, privacy concerns, legal constraints) are employed by SDCs. The strategy which is used by a participant may depend on what the others are using and vice versa. We summarize six key characteristics of the decentralized cloud as follows:

1) Decentralization. Each participant independently makes its own decision on the amount of resources to contribute based on its demand patterns and strategies. There is no central authority to account or monitor their behaviors. Therefore, each participant is free to decline the resource requests from other participants in terms of their strategies or preferences.

2) Self-organization. In the decentralized cloud, self organization is emerging because of service requirements on availability, elasticity, legal constraints and location proximity. The participants can voluntarily build cooperation networks to meet their demands.

3) Geographical distribution. In contrast with large-scale data centers which are merely confined to a few locations, SDCs have a more widely geographical distribution. Consequently, the decentralized cloud will play an active role in delivering services of low latency due to wide location coverage.

4) Heterogeneity. Since resources are provided by different SDCs, they present great heterogeneity in both resource types (e.g., CPU, memory and network) and demand patterns (e.g., availability and prices). Such heterogeneity poses challenges for efficient resource allocation.

5) Interoperability. Interoperability between SDCs is of vital importance for the aggregation of diverse resources in a decentralized manner. In the context of decentralized clouds, the interoperability is the capability of different resource providers to understand each other's service requests, configurations, authentications and application programming interfaces (APIs), in order to collaborate and interoperate with each other.

6) Privacy and security. An immediate benefit of decentralization is improved privacy and security since there is no central control over the data.

Through participating in the decentralized cloud, not only will small cloud players get a channel to form business groups and support each other, but also cloud end-users will gain more confidence in using

this model because of new features. Indeed, the decentralized cloud combines the advantages of both cloud computing and Peer-to-Peer (P2P) computing paradigms. However, there are three key differences between the decentralized cloud and P2P model [6]. First, the peers in the decentralized cloud are mainly small data centers which are legal and economical social entities rather than individuals. Thus, the incentives why cloud providers develop cooperation with each other are not merely for the revenue maximization but also for other social dynamics such as legal constraints, location proximity, trust and loyalty. Second, compared to a large population of peers, the number of cloud providers is relatively small. Third, the computing environment is not so highly dynamic as that in P2P model. Compared to continuously joining and leaving of peers in the P2P system, the participation of SDCs is more stable in decentralized clouds.

It is worthy to mentioning that the decentralized cloud is not going to replace the centralized cloud but to complement each other. Centralized clouds have already demonstrate their power in web hosting and compute-intensive applications. But if applications need high information privacy or better control over the data, the decentralized cloud is the best choice. In the next section, we will further discuss different types of multicloud.

2.1.2 Different Types of Multicloud

From the high level view of architecture, the multicloud model can be categorized into centralized or decentralized type. In this section, we will further subdivide multicloud models into four different types from the perspective of business requirements. We focus on explaining how those multicloud models complement to the existing centralized cloud computing model, *i.e.*, how they serve diverse and evolving business demands from users. The first three multicloud models, namely geographically distributed cloud, hybrid cloud and federated cloud, have centralized architecture while the fourth one, *i.e.*, decentralized cloud, is decentralized cloud architecture.

Geographically distributed cloud. Big cloud vendors such as Amazon, Google and Microsoft have built up massive data centers all over the world. By leveraging location diversity, distributed clouds can enable lower service latency, better load balance and better performance (*e.g.*, service reliability and availability) for cloud services. For example, online content service providers Netflix deploys their services in Amazon distributed clouds so that users can stream Netflix videos from anywhere in the world quickly [4]. In essence, the geographically distributed cloud is a horizontal extension of traditional centralized cloud. All the resources are still owned by the same entity. Therefore, their users still suffer the issues in the data security, privacy and vendor lock-in problem.

Hybrid cloud. There is a great amount of companies, especially large companies, which still own and run their own private data centers for many various reasons. Among which, the most prominent one is the concern on data security and privacy. However, compared to massive cloud data centers, their data centers are relatively small and less efficient in resource elasticity. Hence, they prefer to employ a hybrid compute model where they deploy their core services and data in their private data centers while other less important workload in the public cloud. In many literature, it is also termed as *cloud bursting* that migrates the bursting workload to the cloud. A good example of hybrid cloud model is Dropbox [11] which stores the metadata of the file system in its private data center and stores their users' data in the cloud after encryption. With the help of hybrid cloud, enterprises can use own private data centers for critical services while sending non-mission-critical workloads to the public clouds.

Federated cloud. Another way to expand the capacity of small data centers is federation which aggregates resources from multiple resource providers into a single resource pool. Different from distributed and hybrid cloud models, each participant in the federated cloud is both resource providers and consumers. All participants in the federated cloud trust each other. The resources in the federated cloud are managed by a centralized entity which is recognized by all participants in this federation. The centralized entity also implements a layer of cloud interoperability which allows participants to request and allocate resources for each other. Through cloud federation, each participant can obtain more capacity without investing more hardware. Another similar term to describe a federated cloud is *"cloud of clouds"* in which the resource providers are a combination of public commercial clouds. But in the *cloud of clouds*, each cloud is only resource providers without consuming resources from others. The main purpose of *cloud of clouds* is to add diversity to resource providers so that it can overcome the limitations of single cloud model.

Decentralized cloud. The resources in a decentralized cloud are contributed and shared by a group of individuals or small organizations in a peer-to-peer manner. Similar to the federated cloud, each peer in the decentralized cloud is both resource providers and consumers. But peers in the decentralized cloud have more democratic rights to join or leave the cloud according to their own demands. In addition, all participants can make their decisions on resource allocation in a strategic setting where various strategies based on different preferences (e.g., location proximity, privacy concerns, legal constraints) are employed. A good example of decentralized cloud is SpotCloud [10] (now is acquired by EMC) which provides an online marketplace bringing together resource buyers and sellers in a cloud. Resource providers can monetize their idle resource to increase their revenue while end-users can also benefit from choosing resources from a diverse set of providers based on performance, cost and location parameters.

2.1.3 Resource Allocation in the Multicloud

2.1.3.1 Interacting Entities

There are three interacting entities in the multicloud resource allocation, namely *resource provider, service provider* and *end-user*. Resource provider is the owner of the infrastructure which offers different type of resources, *i.e.*, compute, storage and network. These resources are delivered to service providers in terms of virtualization technologies such as *virtual machine* or *containers* with different Quality of Service (QoS) and pricing models. Service provider rents resources from resource providers and further deploys cloud services that will be consumed by end-users. In return, the end-users use the services that are provided by service providers, generating resource demands. Existing work on resource allocation in the multicloud mainly focuses on how to efficiently allocate resource from the resource provider to the service provider.

There are many interactions among three entities, such as renting/leasing resources among resource providers, transferring data from one site to another and storing service data to resource providers. In this thesis, we will focus on three interactions, namely *cooperation*, *optimization* and *sharing*. Figure 2.1 presents the related work of our thesis.

2.1.3.2 Cooperation, Optimization and Sharing

As it is shown in the Figure 2.1, the first interaction that we discuss is cooperation. The cooperation is the fundamental assumption for optimization and sharing in the multicloud. Resource providers are

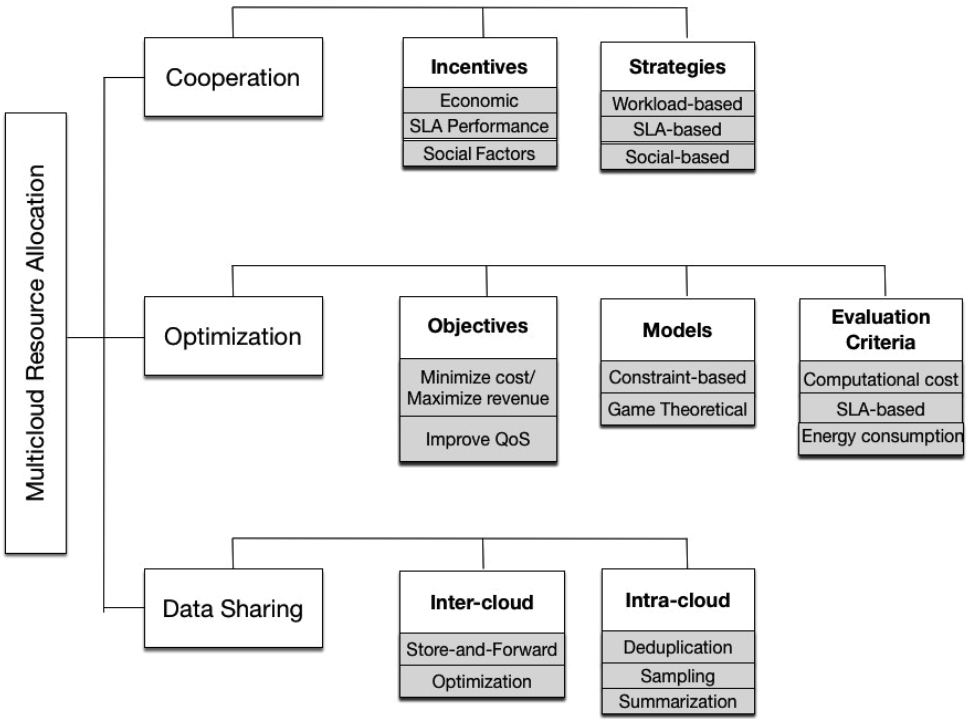

Figure 2.1: Literature review of this thesis

not angels, and they all have incentives to be selfish, *e.g.*, maximize their own revenue or other utilities. Most of existing work assume that each resource provider is cooperative to maximize total social welfare. This assumption usually does not hold in a real world in which different resource providers have diverse resource capacity, workload patterns and resource allocation strategies. Thus, we need to further investigate various cooperation incentives as well as resource allocation strategies. In this way, we can discover that how to develop cooperation in a highly heterogeneous multicloud computing environment.

Second, the efficiency of cooperation can be greatly enhanced by optimization in the multicloud. With leveraging the multicloud, service providers can achieve different service goals, such as minimizing service latency or maximizing network throughput. Mathematically, an optimization model in the multicloud consists of an objective function and associated constraints on the variables. Existing work on optimization in the multicloud involve all resource dimensions including compute, storage and network and various SLAs such as availability, consistency, response time and network throughput. We will further illustrate these optimization work in Section 2.3 in terms of three aspects, namely objectives, models and evaluation criteria.

Finally, we will discuss data sharing in the multicloud. Resource providers in the multicloud are located all over the world and the services in the multicloud are usually deployed globally. The immediate challenge for global services in the multcloud is how to transfer their data in a cost-effective way. Existing work to overcome this challenge includes optimization on inter-cloud data transfer scheduling

with considering different bandwidth price and capacity across different data centers, or intra-cloud optimization on reducing total size of data that are transferred over the network via data deduplication and compression. With these optimization, our data sharing among different data centers can achieve great savings in both transfer time and bandwidth cost.

2.2 Cooperation

What's the conditions of cooperation? If we have a central authority, it is not difficult to develop cooperation for two reasons. First, there is no need to resolve incentives for each participant since all participants in a centralized cooperation aim to maximize social welfare rather than individual utility. Second, it is possible to find an optimal or near-optimal resource allocation strategy since all resources are managed by a central entity. In other words, resource allocation strategies are not that diverse and each participant just follows coordination from the central authority.

What if there is no central authority in the multicloud, under what conditions will cooperation emerge? In this section, we only focus on discussing the cooperation in a decentralized cloud. We firstly give our motivation why we believe the decentralized cloud is important in the future. Then we will discuss the cooperation in the decentralized cloud from two dimensions: incentives and strategies. We analyze the reasons why participants join the decentralized cloud to cooperate with others and how they allocate their resources based on both social and nonsocial factors.

2.2.1 Sharing Economy in Cloud Computing

> *The world's largest taxi firm, Uber, owns no cars. The world's largest accommodation provider, Airbnb, owns no property. And the world's largest cloud provider, Decentralized Cloud, owns no data center.*

The P2P computing paradigm is not successful due to lack of effective business model. For example, in the BitTorrent application, users usually stop sharing their bandwidth after they finish downloading. There is no effective way to incentive users to contribute. Things have changed in the recent years due to the advent of decentralized digital currency. Using a new digital currency for a decentralized cloud provides strong incentive for resource providers to share their resources. A good example is Storj [12] which employs a BitCoin-like transaction system for users to pay each other for resource usage. In the Storj, it is more like a marketplace where users can buy or sell their unused disk spaces. Based on blockchain technology, they can provide users with a decentralized, dependable and privacy-preserving storage service with a good business model. Another example is SpotCloud which also allows customers to sell their idle resources to offer cloud services collaboratively.

In the academia, we also observe new changes on decentralizing the cloud. Chandra et al.[13] first proposed a decentralized edge cloud that uses volunteer edge resources. Apparently, their solution based on volunteers cannot solve incentive problems in the decentralized cloud. Lopez et al.[14], presented their envisions on edge-centric computing. This position paper argues that the centralized clouds should not be the final paradigm, and that a new decentralization wave is necessary. Crowcroft [1] gives us a guideline to build a decentralized and privacy-preserving system with the help of digital payment and privacy protection techniques. There are also a growing body of literature on building a dependable, secure decentralized cloud based on blockchain technologies [12, 15]. Thus, we can envision that in

the near future, more decentralized clouds will emerge from the cloud infrastructure and platform to decentralized services.

2.2.2 Cooperation Incentives

The decentralized digital currency system alone cannot bring success to the decentralized cloud. We still need to motivate resource providers to participate in the decentralized cloud by many other incentives. For example, in the SpotCloud platform, the incentives of resource providers are highly diverse. Some participants may achieve time and cost savings while others may greatly improve their service performance, *e.g.*, response time, data privacy and failure tolerance, through using resources from the decentralized cloud.

There are also many literature on studying economic incentives. For example, existing work [16, 17, 18] all assume that cloud providers in a multicloud environment are with full rationality and the incentives of cooperation are mainly to reduce costs or maximize revenues. Those existing work are imbued in utility maximization with full rationality optimization framework which is a useful idealization used by economists to analyze complex situations. However, it has often been argued that the assumptions entailed in this framework do not hold in real life scenarios [3]. For example, in the real world, people and companies can collaborate simply by friendship, loyalty or trust. Therefore, in order to model decentralized cloud scenario more realistically, we can further explore more social dimensions which have an effect on resource allocation. For example, we will incorporate *social dynamics* such as trade barriers, loyalty, trust and altruism among the CPs and present an analytical model in the decentralized cloud scenario. In addition, we could further add other factors such as biased preferences of different CPs, legal constraints, geographical proximity, data privacy constraints, etc. Through understanding resource providers' incentives, we could design a better business model as well as improve the efficiency of resource allocation.

2.2.3 Resource Allocation Strategies

As the resource providers in the decentralized cloud are of high heterogeneity, their strategies on resource allocation vary greatly. In addition, all resource providers make their decisions on resource allocation in a strategic setting. In other words, the strategy which is used by one resource provider may depend on what the others are using and vice versa. Therefore, it is interesting to ask whether there is a best strategy that outperforms others in the multicloud. Existing work on resource allocation strategies can be divided into three categories, namely based on workload, SLA and social factors, respectively.

Workload-based Strategy. The workload arrival pattern of a resource provider is the main factor that influences its resource allocation strategies. The variations in workload of different resource providers provide opportunities for cooperation. Put simply, resource providers would offer the spare capacity during low utilization periods while seeking resources from others in their peak demands [16]. A typical example is Amazon EC2 which employs three different price models for *on-demand, reserved* and *spot* instances. Thus, a service provider can reduce their cost by replacing expensive on-demand instance with cheaper reserved instance. Toosi et al.[19] present their resource allocation strategies based on exploiting different pricing models.

Another type of resource allocation based on workload is based on heuristic. They make different strategies simply by heuristics that can help reduce cost. For example, Petri et al.[17] discuss three

strategies for exchanging resources in a federated cloud. The basic strategy is that each provider attempts to maximize its revenue from remote tasks without excessively compromising the local tasks on the system. They design different strategies for local and remote tasks, respectively. For local task, the system always accepts the local task without rejection while for remote task, it will be rejected when the requested provider cannot meet the deadline and when accepting remote task can affect the processing of local tasks.

On the other hand, there are also many work on resource allocation based on sophisticated workload modeling. Those work usually model their workload based on queueing theory and stochastic process. In [8], Nancy presents an economical model for resource sharing in a federation of selfish cloud providers(CPs). The workload is based on Markovian model which can model the transitions of the CPs' idle resources. All the CPs in the federation can share their unused resources during the periods of low-demands and borrow spare capacities during the peaks to maximize their revenues. Pal et al.[20], develop their economical models for multicloud service markets with modeling workload arrival process as a M/M/1 queuing system.

SLA-based strategy. SLAs are natural properties of a workload and are equally important for designing resource allocation strategies. The process of resource allocation is the process of assigning enough resources to meet the SLA requirements of a workload. When it comes to optimization, this process is reformulated as an optimization problem with SLA constraints. A common list of SLAs in the cloud includes availability, maximum/mean response time, consistency, data security, location, etc. There are many research efforts on searching optimal resource allocation strategies to meet SLA requirements. We take a multi-cloud storage system as a case study. RACS [21] improves storage diversity by applying RAID-like techniques into multicloud storage, which allows customers to avoid vendor lock-in and single cloud of failure. DepSky [3] further designs two multi-cloud storage protocols to enhance data availability and confidentiality, respectively. SpanStore [22] focuses on reducing latencies for end-users as well as reducing cost for applications by designing different replication policies with considering replication cost, latency and consistency level in the multicloud. All resource allocation strategies in those work are designed based on SLA requirements of cloud services, to name but a few.

Social-based strategy: Social factors also have a great impact on resource allocation strategies. Similar to our social life, a service provider may prefer to share their resources with others in the same country or with others who are friends. The cooperation in Human society is not merely for the revenue maximization, but also for other social factors such as altruism, reciprocity, selfishness, trust, loyalty, friendship, partnership, location proximity and legal constraints. A large body of existing work design their resource allocation strategies based on volunteer resources [13, 23] in which volunteer is essentially a reflection of altruism. Most of the existing work avoid incentive problems in the multicloud with the assumption that resource providers are already cooperative based on reciprocity or centralized power while some of them take selfishness of resource providers into account [16, 24, 25]. Without pre-established cooperation agreements, multicloud framework shall address the issues of trust, legal policy and data privacy [26]. Mihailescu [27] demonstrate that if users become untruthful to report their demands, the *spot* pricing scheme introduced by Amazon EC2 will not be suitable in a federated cloud. There are also some work on P2P cloud [28, 29] which provides a multicloud platform for resource providers to cooperate in a peer-to-peer manner. However, these work is not promising since they do not solve incentive problems in P2P computing. We observe that a system with social-based strategy may not result in a

good business model but a system with a good business model cannot neglect the impact of social factors on the resource allocation.

2.3 Optimization

Optimization in the multicloud can improve the efficiency of cooperation, which can further strengthen cooperative partnership among different resource providers in the multicloud. If the optimizer has full information of all resource providers, the optimization is centralized. Otherwise, the optimization is decentralized since the optimizer only has partial information. For each optimization model, there is an objective function and a set of constraints. In the following, we will discuss various optimization models from the perspectives of objectives, models and evaluation criteria.

2.3.1 Objectives

Many works have been proposed in both academia [3, 22, 30, 31] and industry [32, 33, 34] for using multiple cloud providers to optimize their services. These work show that deploying services over multiple resource providers can improve their performance in terms of different dimensions, such as the cost, network latency, service response time and vendor lock-in. In the following, we will further discuss these objectives.

Minimize cost / maximize revenue. A large body of literature focuses on minimizing the cost, *i.e.*, maximize the revenue, in the multicloud. Samaan [16] presents an economical model for resource sharing in a federation of selfish cloud providers to maximize the revenue of cloud providers. Wang et al.[18] propose a cloud brokerage service that reserves a large pool of instances from cloud providers and serves users with price discounts. It exploits both pricing discounts of long-term instance reservations and multiplexing gains. For example, the billing cycle of cloud providers is normally in hour and partial usage of a billing cycle always incurs a full-cycle charge. Hence, the broker can coordinate users to share the instance-hour in a multiplex way, thereby reducing service cost. They design a dynamic programming formulation for the resource reservation problem with the objective of minimizing service cost. SPANStore [22] minimizes the monetary storage cost by exploiting pricing discrepancies across multiple cloud providers. Their optimization model finds out the suitable replication policy with considering three aspects of information including inter-datacenter latency, application latency, consistency requirements and workload characterization. By replicating data with the best policy, it reduces the storage cost greatly. Liu [35] develops an online resource allocation algorithm to maximize net profits for cloud service providers, which takes advantages of multi-electricity market. Specifically, the electricity prices at different geographic distributed data centers vary over the time throughout a day. This gives an opportunities to reduce the cost of electricity by selecting suitable cloud providers. PostCard et al.[36] minimize the operational costs on inter-datacenter traffic by leveraging the store-and-forward strategies. Thus, their work focuses on determining when the data should be forwarded from a source to a destination or be stored at an intermediate node.

Improve QoS. The other major aspect of optimization in the multicloud is to improve the QoS of multicloud applications. The QoS factors include service latency, network throughput, data consistency, service availability, data availability, vendor lock-in, service independence [37], etc. We summarize the existing work in terms of their objectives as follows.

Service Latency. Choy et al.[38], demonstrates the centralized cloud cannot meet latency requirements of cloud gaming applications. Then they study that how to leverage edge cloud to reduce the service latency. The resources in the edge cloud are from different end-hosts in a more diverse location but are managed with a centralized coordinator. Through application placement and peer selection algorithms, the edge cloud can reduce latency significantly. Bonvin [39] minimizes the communication cost during replicating data among the data centers in different locations through adaptive data placement optimization. They design a virtual economy, where each data partition acts as individual optimizer and chooses whether to migrate, replicate or remove itself with considering its utility and maintenance costs.

Data Avaialibility. Bonvin [39, 40] maximizes data availability by replicating data in the geo-distributed data centers. The availability is modeled as the sum of diversity of each different pair of servers, where the diversity of a pair of servers considers detailed server location in terms of continent, country, data center and room rack. The first model in DepSky [3] DEPSKY-A, improves the data availability by replicating data on several cloud providers using *quorum* techniques. The availability is guaranteed because the data is stored in a quorum of at least $n - f$ clouds and is assumed that at most f clouds can be faulty. Thus, if the number of faulty cloud, *i.e.*, corrupting the data, is less than f, the data is always available.

Network Throughput. The objective of maximizing the network throughput is the same with minimizing the completion time. Meanwhile, high network throughput also incurs high cost on network transfers. Existing work often makes tradeoff between transfer cost and completion time. Wang et al.[41] propose an instance recommendation model for cloud providers to help cloud end-users select instances in SpotCloud. The resource recommendation model targets content distribution applications, which is formulated as a maximum flow minimum cost problem. Amoeba in [42] considers the tradeoffs between bandwidth cost and transfer time. It provides a transfer service for data centers to ensure the completion within deadlines while maintaining high network utilization. The objective in [43] is to maximize the overall weight of all transfer jobs, where the weight of each job is modeled as the urgency level. Specifically, they study that how to optimize data transfer among data centers with considering different urgency level, in order to fully utilize the available inter-datacenter bandwidth.

Energy-efficient Resource Allocation. The multicloud computing environment is of high heterogeneity, which gives many opportunities for optimization. Most of existing work on exploiting resource heterogeneity to reduce energy consumption are based on the idea that scheduling workloads to the energy-efficient server machines (*i.e.*, low power but high performance) [44, 45]. Yigitbasi et al.show that low power Intel Atom processors and high performance Intel Sandy Bridge processors are more energy efficient for I/O bound workloads and CPU bound workloads, respectively. Through exploiting this heterogeneity in a Hadoop cluster, they achieve up to 27% better energy efficiency. On the other hand, Zhang et al.[46] exploit the heterogeneity in both server machine and workload. Based on predicting on the future arrival workloads, they switch on/off the server machines in the cluster, thereby reducing the total energy consumption.

2.3.2 Models

The optimization model in the multicloud can be summarized into two aspects: constraint-based and game theoretical models.

Constraint-based models. Constrained optimization is the process of optimizing an objective function with respect to constraints on model variables. We discuss the objectives in the last section and now we introduce several types of constraints in the multicloud as follows:

- *Resource capacity constraint*: the total allocated resource must be less than the total resource capacity. The resource can be either compute [16, 23, 35] or network bandwidth [41, 47].
- *Budget/cost constraint*: in the real scenario, there is usually a total budget on the resource allocation [23, 41, 48].
- *Deadline/time constraint:* the third popular constraint is time constraint on executing the workload in the cloud [42, 49, 50].
- *Commitment constraint*: it captures the incentives of resource providers to participate in the multicloud. For example, the commitment constraint is that the long term revenue is at least equal to that obtained without participating in the multicloud [16].
- *Regulatory constraint*: in [47], their optimization model incorporates the new regulatory constraint on data placement due to data privacy concerns. For example, Germany data may not be allowed to leave German data centers.
- *Reliability constraint*: it is proposed by [51], which is modeled as the failure probability of all clouds. This constraint can help clients select reliable cloud providers and avoid correlated failures of different cloud providers.

Game-theoretical models. Game theory is mainly used in economics and politics to analyze the conflict and cooperation between different decision-makers. In the multicloud scenario, we have different entities including resource providers, service providers and end-users. The resource allocation in the multicloud can be modeled as a game among different entities and the solution of a game is an equilibrium. There are quite many research efforts on modeling resource allocation in the multicloud using game-theoretical models. The authors in[20] discuss three non-cooperative game theoretical models, which take both price and quality of service (QoS) into consideration, for cloud service markets. Specifically, they consider a monopolistic market with a few big cloud vendors such as Amazon, Google, Microsoft and focus on the non-cooperative game analysis of price and QoS competitions among these major vendors. Li [25] designs algorithms for inter-cloud resource trading and scheduling in a federation of selfish clouds. Their work aims to maximize net profit of each participant using a double auction-based approach. The model considers SLA constraints in the service latency. Their model preserves two important properties: 1) truthfulness: the dominant strategy is to bid true valuations; ii) individual rationality: each cloud obtains a non-negative profit by participating in the auction. Jalaparti [52] proposes Cloud Resource Allocation Games(CRAGs) to model the interactions among different entities in the multicloud. A CRAG is a non-cooperative game in which cloud end-users selfishly maximize their utility. In addition, they discusses a variant of CRAGs called Stacklberg CRAGs which captures the interactions between cloud providers and service providers. Niyato [53] studies resource and revenue sharing in the multicloud using a hierarchical cooperative game model. Specifically, they solve one key problem of sharing resources and revenues in a coalition. They analyze the stability of a coalition in

which no individual cloud has an incentive to change the decision on resource allocation, since the individual profit of all providers is maximized. They further find the optimal coalition that maximize the total profit instead of individual one. Their results show that forming coalition in a multicloud leads to the higher profit. Coalitional game theory is also applied to solve the problems, *e.g.*, how to share the profit generated by federation [54] or how to reduce energy consumption through coalition formulation in the multicloud [55].

2.3.3 Evaluation Criteria

Finally, we summarize a list of criteria which are used to evaluate the performance of different optimization models. The performance of model can be evaluated from two perspectives. First, from the solver perspective, we can evaluate how many resources are needed to solve the model, such as CPU and memory usage, computational cost, etc. Second, from the solution perspective, we can evaluate the goodness of the solution that is searched by optimization model. In the following, we list these criteria as follows:

- Computational cost: it is mainly the amount of time that needs to solve the model [50]. For example, the computational cost of a constrained model is mainly determined by the number of variables and constraints. The *average allocation time* in [42] or *allocation delay* are the same concepts with different names, which evaluate the time cost of solving the mode.
- Profit/revenue/cost/gain: those are the most common criteria to evaluate the performance of resource allocation [16] [19], [17] [35]. The cost in the multicloud includes compute, storage and bandwidth cost while the revenue is made from serving user's requests or resource trading among resource providers in the multicloud.
- Average latency/delay or response time: those are commonly used for online cloud services which normally are the sum of processing and network delay. For example, Depsky [3] evaluates the average latency of reads and writes in multicloud storage system.
- Network throughput/utilization: thoes are widely used for evaluating network performance of data transfer system [42] or multicloud storage systems[3].
- Availability: it can be measured for both data and cloud services. Data availability in [3] is measured as the ratio between the total number of success reads and the total number of tried reads while the service availability is measured as the service up time.
- Load balance: it is measured in terms of the number of requests that is dispatched to different data centers [35, 56].
- SLA violations: service-level agreements (SLAs) are defined to evaluate the performance of different cloud providers. The less the number of SLA violations is, the better the cloud provider is. Bonvin et al.[57], measure SLA violations in term of response time while [19, 42] evaluates the performance of resource allocation with Request Acceptance/Rejection Rate. In [17], authors measure the reputation of cloud provider as the proportion of tasks that are completed within the deadline.
- Energy consumption: it is to evaluate the energy-efficient resource allocation [44, 45]. It can be also measured as the reduction of energy consumption.

2.4 Data Sharing

Currently, cloud services and applications are becoming increasingly more data intensive. Data are generated distributedly at various hardware (*e.g.*, cameras, mobile phones, sensors, etc.) and software components (*e.g.*, log system). Moving large amounts of distributed data into the cloud or from one cloud to another can incur high costs in both time and bandwidth. It is a challenging task to share these big data in a cost-effective way. First, the size of data is unprecedentedly big, compared to limited bandwidth resources. Second, traditional protocols such as FTP and HTTP for moving big data over the Internet are highly inefficient. Third, the bandwidth cost on Internet transfer is high. If the size of data is big enough, the bandwidth cost is even much higher than shipping the disk directly. The optimization on data sharing in the multicloud can be summarized into two aspects: inter-cloud data transfer scheduling and intra-cloud optimization. The former focuses on how to reduce data transfer cost with an optimized scheduler while the later discusses the related work on how to reduce the size of big data and on how to design big data transfer protocols. We detail existing work as follows.

2.4.1 Inter-cloud Scheduling

Many optimization algorithms and strategies have been proposed to schedule data transfers among data centers with different objectives in the literature. The store-and-forward strategy adopted by [36, 58, 59, 60] is effective for delay tolerant bulk data transfer, which reduces the cost by scheduling data transfer at off-peak hours [36, 58]. To increase the throughput and thus minimize the transfer time, StorkCloud [61] integrates multi-protocol transfers aiming at optimizing the end-to-end throughput while considering the link capacity, disk rate and the CPU capacity. NetStitchter [59] employs a network of storage nodes to schedule the data transfer in a store-and-forward manner based on predictions on the available leftover bandwidth at access and backbone links. In order to decrease the cost, Postcard [36] formulates an online optimization problem to minimize costs on inter-datacenter traffic by exploiting the price discrepancy in different locations. Other works such as [62] designs a set of transfer strategies to readjust to different network conditions (e.g., network latency, available bandwidth) between cloud data centers whereas [63] focuses on the evaluation of different data transfer protocols for big data. Wu *et al.* [43] proposes scheduling approaches for bulk data transfers with different urgency levels. There are also many work on managing the tradeoff between time and cost. Existing work presented in [64] and [65] optimized the time and cost performance for online services (e.g., search engine) and content multihoming (e.g., CDNs). Tudoran *et.al* [66] proposes transfer as a service for multi-site cloud with considering the cost in both computation and network.

2.4.2 Intra-cloud Optimization

Besides scheduling data transfer over the network among different resource providers, we can also conduct optimization before the data is transferred over the network. In the following, we will discuss two classes of optimization on making big data smaller and designing new protocols for big data transfer, respectively.

Making big data smaller. There are several approaches to making big data smaller, which can be classified into two categories. The first category is information lossless methods such as data compression and data deduplication, which can preserve all the information in the original dataset. Data com-

pression is suitable for all kinds of data, which reduces the total size of data by different compression algorithms, *e.g.*, Lempel–Ziv and Huffman encoding. RainStor, [67], develops a proprietary compression algorithm that promises to squeeze data into one fortieth their original size. In addition, they can even support standard SQL query over their compressed data. Data deduplication reduces the total size of data by avoiding retransmission the duplicate copies of repeating data. It employs fingerprinting algorithms such as SHA-1. MD5 to detect the duplicates in the files. Hence, data deduplication is very effective for the data with high duplicates, *e.g.*, Web data or operating system image files.

The second category is lossy methods such as data sampling and data summarization, which lead to information loss. Data sampling is a simple yet effective way to return a small random subset of selected data. This method shall guarantee that the sample is representative of the distribution which governs the original data. The major sampling techniques include simple random sampling, stratified sampling, quota sampling, etc [68]. Data summarization aims to identify a useful subset of the original dataset. The 'usefulness' is defined based on data properties such as representativeness[69, 70], diversity [71], informativeness [72] and coverage [73]. In this way, data summarization can minimize the information loss, *i.e.*, preserving more information in the original dataset. In the real scenario, we can combine methods in both categories to further reduce the size of data. For example, before the data is sent to the network, we can firstly sample a subset from the original data, then remove duplicates and finally compress the data.

Designing big data transfer protocol. The ubiquitous Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are not designed for big data transfer, which have performance issues over long-distance bulk data transfer. There are quite many work on improving current protocols for big data transfer, including high-speed TCP, FAST TCP, UDT, etc. There are also active efforts from the industry. Aspera [74], now acquired by IBM, designed its FASP protocol which combines the advantages of both TCP and UDP protocols. FASP protocol uses one TCP port for session initialization and control, and one User Datagram Protocol (UDP) port for data transfer. As a result, FASP transfers can maximize the utilization of available bandwidth to provide a guaranteed delivery time regardless of file size, transfer distance or network conditions. CloudOpt [75] helps companies dramatically accelerate the movement of data to and from the cloud through applying compression, deduplication, and protocol optimizations. It accelerates most TCP protocols including, HTTP, MySQL, SCP/SSH, FTP, NFS, and CIFS. Cloudbeam [76] provides full solution for local-to-cloud replication, and cloud data migration. Its Managed File Transfer (MFT) uses a proprietary transfer protocol that supports multi-part transfers, concurrent sessions, streams, SSL encryption and block-level recovery. It shows that their performance is 10-12X over standard copy [77].

2.5 Related Project

Scalia

Scalia [31] is a cloud storage brokerage solution that employs multiple public storage clouds to increase data availability and durability as well as to avoid vendor lock-in. The system adapts the placement of data across different clouds based on data access patterns. Specifically, it supports multiple optimization objectives, *e.g.*, minimizing the cloud storage cost given a set of customer constraints such as availability, durability; maintaining a certain monthly budget by relaxing some constraints, such as vendor lock-in or availability. It also supports incorporating user-defined constraints on data durability, availability and

level of vendor lock-in. From the architecture point of view, it consists of three layers: 1) engine layer to compute the best provider set in terms of user requirements based on access pattern of the data, and to distribute and store the data chunks at the most suitable cloud; 2) cache layer to improve the performance for frequently read data and reduce the communication cost; 3) database layer for hosting the metadata and access history of the data.

DepSky

Bessani et al.[3] propose a multicloud system which utilizes four public clouds. They design two storage protocols that improve the availability, integrity and confidentiality of information stored in the cloud through the encryption, encoding and replication of the data in the multicloud. They use the term *cloud of clouds* to describe their multicloud environment as a virtual cloud of diverse commercial clouds. DepSky enables cloud interoperability by a three abstraction level data model, which encapsulates the heterogeneity of the interfaces of different cloud providers. Through leveraging resources from different clouds, their system can overcome vendor lock-in problems and improve data availability and privacy.

NCCloud

NCCloud [30] implements network coding algorithms in a multicloud which aims to provide fault tolerance for cloud storage. Their work focuses on implementing traditional network coding-based storage system *i.e.*, regenerating codes, in a multicloud environment. The key issue is how to restore the lost data in one cloud from other surviving clouds in a cost-effective manner. NCCloud implements a practical version of the functional minimum storage regenerating code (F-MSR), which regenerates new chunks during repair subject to the required degree of data redundancy. Comparing to the traditional erasure coding schemes based on RAID-6, the storage cost of F-MSR is the same but F-MSR uses less repair traffic when recovering a single-cloud failure. To sum up, NCCloud is effective to avoid single cloud failure and suitable for rarely-read long-term archival applications.

SpanStore

While most of the works focus on using multicloud to improve availability, durability, vendor lock-in, and consistency, SpanStore [22] focuses on the cost optimization in a multicloud environment. SPANStore provides geo-replicated storage service across multiple clouds. Data replication is a common practice to maintain high availability and reduce access latency at the cost of more storage space. Hence, it is important to determine where and how many objects are replicated. SpanStore firstly minimizes the monetary cost by exploiting pricing discrepancies across multiple cloud providers. Second, the system determines the suitable replication policies with considering latency and consistency constraints. Finally, SPANStore further reduces cost by minimizing the necessary compute resources to offer a global view of storage.

RESERVOIR

Resources and Services Virtualization without Barriers (RESERVOIR) [7] aims to create the next-generation cloud computing platform, which can enable multiple independent providers to cooperate seamlessly to maximize their mutual benefits. In this way, the architecture of Reservoir is peer-to-peer rather than centralized. Different cloud providers can communicate directly with each other and share their resources. They define their decentralized peer-to-peer cloud as a federated cloud. Through federation and interoperability, Reservoir can aggregate resources to provide a seemingly infinite resources. In particular, unlike other multicloud models with big cloud vendors only, many small and medium-sized

enterprises (SMEs) can become cloud providers to monetary their idle resources in Reservoir. Each resource provider in Reservoir is an autonomous entity with its own goals. All providers can choose their partners according to their own preferences (*e.g.*, location proximity, privacy regulations and SLAs). Reservoir provides interoperability layers for resource providers to express their business requirements and negotiate resource sharing.

Apache Jclouds

Apache Jclouds [6] is an open source multicloud libraries, which allow developers to create applications that are portable across different clouds. Using Jclouds, developers can only focus on designing their applications while Jclouds enables cloud interoperability for many public cloud providers. Jcloud defines *Views* as portable abstractions that allows developers write code without limited their applications to a specific vendors. It provides compute and storage APIs and supports for nearly 30 cloud providers including Amazon, Azure, GoGrid, OpenStack, Rackspace, and Google. At present, Jclouds is becoming the *de facto* standard library to build portable multicloud services.

CloudSpaces

The CloudSpaces [78] project is funded by the European Framework Programme 7 (FP7), which aims to create the next generation of open personal clouds. In addition, it advocates for a paradigm shift from application-centric models to person-centric models where users can retake the control of their information. To achieve this goal, it consists of three building blocks: CloudSpaces Share, CloudSpaces Storage and CloudSpaces Services. These three parts are corresponding to the following three challenges in the multicloud: 1) how to enable cloud interoperability and privacy-aware data sharing among personal clouds; 2) how to manage data from heterogeneous storage resource in a scalable manner; 3) how to provide a high level service infrastructure for third-party applications that can benefit from personal multicloud models. The project presents StackSync, an open source Personal cloud that includes advanced synchronization, elasticity, security, and interoperability with other cloud platforms. Its interoperability protocol enables folder sharing and collaboration among the heterogeneous personal clouds such as NEC Personal Cloud, OpenStack and Ubuntu one.

SUPERCLOUD

The SUPERCLOUD project [79] aims to develop new security and dependable multicloud infrastructure, which targets for applications in the healthcare domain. Their approach allows users to define their own protection requirements and to avoid vendor lock-ins. Also, they focus on self-managed services for self-protecting multicloud which can reduce administration complexity. Specifically, there are four parts in this project to achieve their goals, namely cloud security, inter-cloud architectures, security protocols and software-defined network. Through their design on architecture, protocol and network, the project provides the self-service security, *i.e.*, users can specify their own protection policies; the self-managed security, *i.e.*, security framework operates seamlessly over compute, storage and network layers; and finally end-to-end security, *i.e.*, the architecture provides trust models and security mechanisms across different cloud service providers. This is an ongoing project which meets the increasing demands on more flexible and manageable cloud security solutions.

Storj

Storj (pronounced: storage) [9] is a decentralized storage cloud that is based on the latest blockchain technology. Unlike centralized cloud storage providers such as Dropbox using storage resources in the big data center, Storj mainly rents storage resource from individuals. It builds an autonomous network

using blockchain technology which allows users to buy and to sell hard drive space. Users' privacy and their data are highly protected thanks to the blockchain technology that using a set of cryptographic algorithms. Compared to renting resources from centralized cloud, Storj claims that the cost of cloud storage can be reduced by orders of magnitude, 10-100x cheaper. In addition, Storj employs a digital currency system Storjcoin X (SJCX) which allows users to purchase storage or earn money from renting free hard disk space. This is an incentive mechanism to encourage more users to participate in the network. To sum up, by offering stronger privacy, security, and proper monetary incentives based on blockchain technology, Storj is on the way to decentralize storage cloud in a privacy-preserving manner.

2.6 Summary

In this chapter, we discuss multicloud resource allocation from the perspectives of cooperation, optimization and sharing. This thesis work includes all those three aspects. For the cooperation in Chapter 3, we propose a new decentralized cloud model and discuss how resource providers collaborate with each other by using various resource allocation strategies. In Chapter 4, we optimize the information leakage in the multicloud storage system. The objective in this work is to reduce the information leakage to each cloud in the multicloud storage system. We also design new metrics to evaluate the effectiveness of our model. For the data sharing, on the one hand, in Chapter 5, we design an optimization algorithm for data sharing among data centers in a peer-to-peer manner. The objective of our optimization is to find a suitable trade-off between the transfer time and the bandwidth cost. On the other hand, we design a data summarization algorithm to reduce the total size of dataset, thereby reducing the total network transfer as well as achieving both time and cost savings. In the following chapters, we will present our work.

Part II

Cooperation

Chapter 3

Decentralizing the Cloud: How Can Small Data Centers Cooperate?

The empire,long divided,must unite;
long united,must divide.

Luo Guanzhong in the Romance of the Three Kingdoms

3.1 Introduction

Cloud computing has become hugely popular in the IT marketplace because of its on-demand resource provisioning, high availability and elasticity. These features allow cloud end-users to access resources in a pay-as-you-go manner and to meet varying demands without upfront resource commitments [4]. Currently, many small data centers (SDCs) are springing up in order to avoid severe issues (e.g., legal, privacy, and data control [80]) that can arise with the adoption of massive centralized data centers. According to recent news reports [81], many European countries are now concerned about their data leaving European borders and are also stressing on the need for *local* small data centers to serve local needs. For instance, in a country like Switzerland with various autonomous cantons, one could imagine the emergence of multiple small data centers in various government institutions to obviate the reliance on big enterprise data centers and to maintain control over local data.

However, these small data centers, due to their size, are likely to suffer from resource under-provisioning thus failing to meet peak demand. We term this the *Resource Provisioning Dilemma* faced by SDCs in that they have to make the tradeoff between request loss and the cost of over-provisioning. One way out of this dilemma for such small data centers is to cooperate with each other to help meet each others' user demand, thereby increasing their resources without having to invest in more. Such cooperation is analogous to *Business Clusters* described in mainstream economics which emerge due to, among other factors, shared interests and geographical proximity [82]. We note here that since one of the key benefits of SDCs is the avoidance of legal and privacy issues, it is apparent that SDCs much like *Business Clusters* would prefer to cooperate with those SDCs which lie under one legislative control, thereby alleviating any of their own or their customers' privacy concerns. An alternative could be a big data center under one legislative control, however in practice due to bureaucratic obstacles and lack of centralized

coordination, most local enterprises such as universities, research centers, hospitals, and govt. offices, etc., often resort to employing their own SDCs. Hurdles in centralizing data control in general due to organizational mismatches and lack of incentives have been formalized in [83].

Some previous works [13, 28] have focused on platform design for cooperation between cloud providers. Other works have focused on prices and revenue maximization for incentivizing data centers to cooperate [16]. However, in a dynamic setting of data centers with varying workload patterns and changing sociopolitical and legal realities, prices and revenue maximization are not the sole (or at times even relevant) criterion that can determine cooperation. Factors such as *location proximity*, *workload similarity*, *privacy concerns* etc., can be equally, if not more, important considerations for resource exchange and allocation between small data centers.

In this chapter, we propose a decentralized cloud with a swarm of networked SDCs which cooperate with each other under varying workloads, and which employ various strategies in order to cooperate with others. Our analysis aims to answer the following questions: 1) what are the incentives of SDCs for cooperation within the decentralized cloud; 2) how is the performance of cooperation affected due to different strategies adopted by SDCs; 3) what type of strategy can thrive in a heterogeneous environment. Specifically, the main contributions of this chapter are summarized as follows:

(1) We propose a model for decentralized cloud with a swarm of networked SDCs. Three workload arrival models are introduced to simulate different levels of workload burstiness in the SDCs.

(2) A general strategy function that can be employed by SDCs is proposed to evaluate the performance of cooperation between the SDCs. Moreover, we design four specific strategy functions based on different dimensions: *capacity, history, prediction* and *reciprocity*, to model conditions under which SDCs can choose to cooperate with each other.

(3) Through extensive simulations of realistic models of data centers, we analyze the performance of various strategies from the perspective of both the decentralized cloud as a whole, and individual SDCs. We discover that the strategy with the best performance is the simplest strategy which is based on reciprocity. Furthermore, we design a new adaptive approach for SDCs to learn how to select a strategy effectively in a heterogeneous environment. The results reveal that most of the SDCs will eventually converge on the same strategy in order to achieve the state of stable mutual cooperation.

3.2 Related Work

Work focusing on decentralizing the cloud includes [13], which proposed decentralized cloud that uses volunteer edge resources. Wang *et al.* [41] study a cloud platform built on customer-provided resources called SpotCloud, which allows customers to sell their idle resources to offer cloud services collaboratively. Other approaches [28] design and implement a decentralized cloud computing platform in an Infrastructure-as-a-Service level, based on P2P technologies, while our work focuses on studying the incentives and strategies of resource sharing among the SDCs and on how the SDCs can cooperate in a decentralized manner.

Our work is not alone in exploring the resource sharing models in the multi-cloud environment. However, most of the previous studies are based on the federated cloud environment which often relies on centralized coordination with limited number of cloud providers [16, 17]. Samaan [16] presents an economical resource sharing model, which is based on Markovian model, in a federation of selfish cloud providers (CPs). However, her model relies on a centralized federation broker which performs the VM

allocations among the cloud providers. The policies in [17] focus on helping providers to make the decisions on executing a job locally or remotely to improve providers' profit. On the other hand, [18] propose resource sharing models based on brokerage services, which assume a centralized broker to aggregate the resource demands and to bridge the gap between the cloud providers and users. Our work is distinct from these in that we do not assume a federated cloud of selfish cloud providers with full rationality and where resource sharing is only based on monetary payment.

The decentralized cloud model introduced in our work combines the advantages of both cloud federation model [7] and peer-to-peer (P2P) model. It is similar to P2P in that it is decentralized and self-organizing for the aggregation of resources to meet user demands. Instead of using monetary payment, our model uses reciprocity-based scheme as an incentive for sharing resources, which is similar to the Tit-for-Tat strategy in file swarming systems [84]. The strategies designed in this chapter are inspired by Axelrod's work on the conditions for the emergence of cooperation [85]. Nevertheless, compared to normal P2P systems, the number of SDCs is much smaller and the computing environment of our model is not so highly dynamic as a P2P environment. Thus, in contrast to P2P, we do not face the problem of churn, and the participation of SDCs is more stable. Also, other factors such as legal concerns for choice of partners can be of importance, which is not the case in P2P. We can design a general strategy function, which incorporates various factors of resource sharing, to evaluate the performance of cooperation. Finally, we are not aware of other works that study the effect of different cooperation strategies on service satisfaction in the decentralized cloud.

3.3 Model Description

A decentralized cloud computing system consists of a swarm of networked small datacenters (SDCs). Each of the SDCs has different capacities, various workload arrival patterns and diverse strategies for resource provisioning. In this section, we will explain the models in this chapter.

3.3.1 Small Data Center

We assume that an SDC has a certain number (hundreds) of networked servers which can host multiple virtual machines (VMs). Each VM requires a set of resources, including CPU, memory and storage to serve the workloads which are submitted by cloud end-users. A *workload* is defined as a resource request from cloud end-users, with variable execution time. The different SDCs have different levels of request burstiness, which can be modeled with various workload arrival models. We assume a discrete time horizon, $t \in \{0, 1, \cdots, T\}$. After receiving workload requests at time t, an SDC will find available resources among all its physical servers, which will be returned as VMs, to serve user requests. If all the servers within a single SDC are busy, the SDC will reject the resource requests, which results in revenue loss and user dissatisfaction. In our cloud model, we assume that the performance, in terms of satisfied requests, of each SDC depends on its workload arrival and free capacity. From the perspective of resource elasticity, the *request loss* of an SDC i is defined as the accumulated amount of under-provisioned resources, which is formulated as:

$$loss_i = \sum_{t=0}^{T} max(d_i(t) - c_i(t), 0) \tag{3.1}$$

where $d_i(t)$ is the number of resource requests and $c_i(t)$ is the free capacity of the SDC i at time t. $d_i(t)$ and $c_i(t)$ vary with time t and the SDC i is under-provisioned when $d_i(t) > c_i(t)$. To minimize the loss of request, the SDC is forced to optimize its capacity and demand planning (e.g., buying more resources or shifting the demand). However, we will later discuss how SDCs can alternatively reduce their request loss through cooperation within the decentralized cloud. It is noted that we do not consider the impact of quality of service (QoS) on the request loss. We assume that VM techniques provide good performance isolation so that the QoS remains unaffected with the fluctuation of requests.

3.3.2 Workload Arrival Models

In this section, we will discuss the workload arrival models based on three types of stochastic arrival processes, namely Poisson process, Markov-modulated Poisson process and heavy-tailed process. These models have been extensively used for modeling real world traces of job arrival in cloud datacenters [86]-[87]. With three different arrival models, we can simulate the SDCs with different levels of workload burstiness and we assume the arrival of workloads at different SDCs is independent from each other, i.e., they are independently sampled. We believe that this is a fair assumption since individual SDCs are independent of each other and can have different sets of customers with different demand patterns.

3.3.2.1 Poisson Arrival

The Poisson arrival process [86, 87] is a counting process which counts the number of events in a given time interval. In our cloud scenario, an event is the request of a VM from the SDC. Thus, the workload arrival process of an SDC, $\{D(t), t \geq 0\}$, is a Poisson process with the mean rate λ if:

$$P[D(t+\tau) - D(t) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!}, k = 0, 1, ...,$$

where k is the number of requests in the time interval $(t, t+\tau]$. The Poisson arrival has a low level of burstiness due to the independent increments property, i.e., the number of requests arriving into the system after time t is independent of the number of requests arriving into the system before time t.

3.3.2.2 Markov-modulated Poisson Process

A Markov-modulated Poisson Process (MMPP) [86]-[87] is a doubly Poisson process whose rate varies according to a Markov process. It is particularly useful in modeling the process with the time-varying arrival rate. We consider a Markov process with state space $\{1, ..., r\}$ where each of the r states corresponds to an arrival rate λ_i, i.e., when the Markov process is in state i, the rate of MMPP is λ_i. In our model, we consider a simple MMPP with two states, namely high (λ_h) and low (λ_l). The transition probabilities between high and low state are denoted as p_h and p_l. Thus, the expected rate of $D(t)$ is $\lambda = \frac{p_h}{p_l+p_h}\lambda_h + \frac{p_l}{p_l+p_h}\lambda_l$. We can control the level of burstiness through adjusting the transition probabilities between the high and low rates.

3.3.2.3 Heavy-tailed Arrival

We consider the case of heavy-tailed arrival [86, 87] in which we model the burstiness of requests as Bounded Pareto distribution. A Bounded Pareto Distribution (BP(L, H, α)) is characterized by 3 parameters L, H and α, where L and H are the low and the upper bound for the number of service requests respectively. The probability density function is defined as:

$$f(x|L,H,\alpha) = \frac{\alpha L^{\alpha}}{1-(\frac{L}{H})^{\alpha}} x^{-(\alpha+1)}, L < x < H$$

where $\alpha > 0$ is a shape parameter which is inversely proportional to the variance of the distribution. In our cloud scenario, H, L can simulate the high peak and low peak of the service requests. The expected request rate is $\lambda = E(X) = \frac{L^{\alpha}}{1-(L/H)^{\alpha}} * \frac{\alpha}{\alpha-1} * (1/L^{\alpha-1} - 1/H^{\alpha-1}), \alpha \neq 1$.

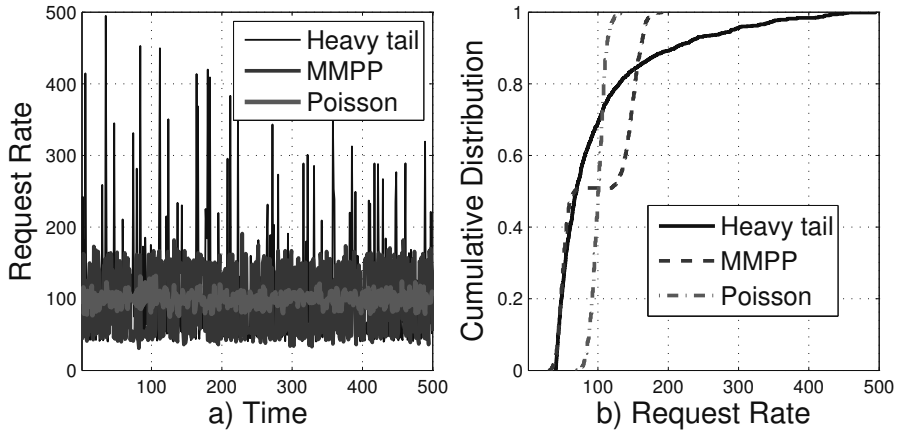

Figure 3.1: Characteristics of the three workload arrival models

***Discussion**:* To compare the three different types of workload arrivals, we set the parameters to have the same mean workload arrival rate. As is shown in Figure 3.1(a), the mean arrival rate of all three workload arrivals is around 100, but they evidence different levels of burstiness. The Poisson arrival with $\lambda = 100$ is the most stable which fluctuates around the average rate while the Heavy-tailed workload varies drastically between the low rate ($L = 40$) and the high rate ($H = 500$). Figure 1(b) shows the cumulative distribution of arrival rate. In this Figure, we can see that MMPP has about 50% of arrival rates around high rate ($\lambda_h = 150$) and the remaining half around low rate ($\lambda_l = 50$) since the transition probabilities are the same ($p_l = p_h = 0.5$). We set $\alpha = 1.2$ which results in about 20% of request rates greater than the average.

Queueing theory has been traditionally used in request arrival and service problems. However, in our three workload models, the service times do not follow the exponential distribution. Thus, we should consider MMPP and Heavy-tailed as two renewal processes while Poisson arrival process in our case is a special case of M/G/n queue. Although some limited analytical derivation for such queue models has been proposed in the literature, their solutions are often mathematically challenging [88]. Taking inspiration from [89], we use a simulation based approach (as opposed to an analytical approach) to evaluate the impact of different workload arrivals.

3.3.3 Decentralized Cloud

We consider a decentralized cloud with n networked SDCs. By participating in the decentralized cloud, SDCs can work collaboratively and support each other, i.e., share their unused capacities during low-demand periods or borrow spare capacities during peaks. In other words, all the SDCs in the decentralized cloud form a resource sharing network which can be represented as a weighted directed graph $G = \langle V, E \rangle$. Let P be the set of all SDCs as the vertices (i.e. $P = V$) and the edges are represented as the cooperation between the SDCs. For example, an edge $e_{ij} \in E$ connects two SDCs $p_i, p_j \in P$ in the direction $i \rightarrow j$. In addition, the weights w on the edges $e \in E$ are given by a *strategy function* $v_e(\cdot)$ which evaluates the cooperation between two SDCs in an edge. Therefore, the weight of an edge connecting two SDCs indicates how well they cooperate together. We do not assume that all the SDCs share the same strategy function since different SDCs may have different ways to evaluate the cooperation with their partners. Instead, we define the strategy function $v_e(\cdot)$ as a *black-box* function which can evaluate cooperation in terms of different dimensions of resource sharing. For example, the strategy function can take factors such as location proximity, network latency and workload arrival patterns into consideration. We will further discuss the strategy function in the next section.

Furthermore, we define the *partners* $\Gamma(p)$ of an SDC $p \in P = V$ as the set of SDCs who can share the resource with p. In our model, we assume the partnership is symmetric (i.e., if $p_j \in \Gamma(p_i)$, then $p_i \in \Gamma(p_j)$) and non-transitive (i.e., if $p_i \in \Gamma(p_j)$ and $p_j \in \Gamma(p_k)$, it does not mean $p_i \in \Gamma(p_k)$). In other words, an SDC cannot share the resources with a partner of its partners. We also assume that resource sharing among the SDCs in the decentralized cloud is free of charge rather than based on monetary payment scheme [90]. All SDCs maintain their local history of the interactions with their partners and use this information to evaluate the performance of cooperation. Furthermore, each SDC makes its own decision independently based on its strategy function and there is no central authority to monitor their behaviors. In the following, we discuss three modes of decentralized cloud:

3.3.3.1 Non-cooperative

In this extreme case, there is no cooperation among different SDCs, i.e., $E = \{\emptyset\}$. Thus, given any SDC p_i, it will not share its resource with anyone else since its partners set $\Gamma(p_i)$ will be empty. Therefore, the total capacity of p_i at any time t remains the same as its initial capacity at the time 0. Due to the lack of cooperation, SDCs in this mode cannot borrow resources from others. They will suffer losses because of failure to increase their limited capacity through cooperation.

3.3.3.2 Fully-cooperative

In the fully-cooperative decentralized cloud, all the SDCs share their resources and support each other. In this case, given any SDC p_i at any time t, it can share the resources with any other SDCs in the decentralized cloud, i.e., partners set $\Gamma(p_i) = \{p_j \in P | j \neq i\}$. Thus, the total capacity of each SDC can change over the time. However, an SDC in the decentralized cloud can still suffer a loss but only when there are no other SDCs with enough free capacity to support it, i.e., $\forall p_j \in P, \nexists c_j(t) > d_i(t)$, where $j \neq i$.

3.3.3.3 Cooperative

In this mode, each SDC has a limited number of partners (i.e., $\Gamma(p) \neq \{\emptyset\}$) and they can share unused resources and support each other. Moreover, each SDC p_i will evaluate the performance of cooperation with its partners via the strategy function, i.e., $v(e_{ij}), p_j \in \Gamma(p_i)$. Through evaluation, p_i can manage its partners set (e.g., replacing current partners with better ones) in order to achieve different goals, varying from minimizing the request loss to improving the service response time. In this chapter, we assume the SDCs initially build the set of partners based on location. We also set the maximum number of partners for each SDC. Thus, the SDC p_i will be under-provisioned only when all the partners in $\Gamma(p_i)$ do not have enough resources, i.e., $\forall p_j \in \Gamma(p_i), \nexists c_j(t) > d_i(t)$.

3.4 Evaluating Cooperation with strategy function

In this section, we discuss the strategy function which evaluates the performance of cooperation between two SDCs. It is a black-box function since no SDC has knowledge of the others' strategy functions. Nevertheless, the strategy function accounts for whatever optimization the SDCs can do on dimensions of the resource sharing. We define a finite set X of resource sharing factors. In addition, given the resource sharing network $G = \langle V, E \rangle$ with edge weights w, the strategy function that is associated with an edge e can be formulated as:

$$w = v_e(\mathbf{x}), e \in E, \mathbf{x} \in X \tag{3.2}$$

where the analytic form of $v_e(\mathbf{x}) : X \rightarrow \mathbb{R}$ is unknown and $\mathbf{x}$ represents the different factors associated with SDCs. For example, $\mathbf{x}$ can be a vector of factors such as capacity, workload pattern and network latency. The weight w measured by strategy function indicates the performance of cooperation, which serves two purposes: resource sharing evaluation (RSE) and partnership management (PM). In turn, RSE can be used for two purposes: lending and borrowing. The objective of lending evaluation is to decide if an SDC agrees to lend resources to others while that of borrowing evaluation is to decide from whom an SDC should borrow. For example, if an SDC p_i needs resources, it will send resource requests to all its partners $\Gamma(p_i)$. Each partner in $\Gamma(p_i)$ will evaluate the performance of potential cooperation with the strategy function $v_{e_{ji}}(x)$ and send the results of evaluation to p_i. After receiving all the responses from its partners, p_i will also evaluate the cooperation with $v_{e_{ij}}(x)$ and select the best one to work collaboratively. Second, as for PM, the strategy function provides the results on the performance of cooperation, which helps the SDCs replace less cooperative partners with more cooperative ones. In the following, we will study four types of strategy functions:

1) General Strategy: The general strategy is the basic one which takes the SDC's capacity into consideration. This strategy only guarantees that an SDC has enough available capacity to offer its help to others. The SDC with the general strategy is too short-sighted or unconcerned to have second thoughts on the impact of its sharing behavior, which makes it more generous than other strategies. Thus, the strategy takes the user generosity into consideration and has been inspired by [90]. Meanwhile, this strategy is the base one for other strategies since having enough resources is the base for cooperation among the SDCs.

2) History-based: In the history-based strategy, the SDCs need to save the history of past interactions. The reliance on history to make cooperation decisions has been utilized extensively in the P2P literature where they are usually described as indirect reciprocity schemes or reputation systems or community

level incentives [91, 92]. We denote the interaction as $r(p_i, p_j, t)$ which indicates the accumulated number of resources that p_i has lent to p_j at time t. Let the global history H be the set of the interactions of all SDCs and H_i is the history saved by SDC p_i. Thus, p_i can assess the behavior of p_j at time t based on

$$H_i^t(p_j) = \{r \in H_i | r = r(p_i, p_j, t) \; or \; r = r(p_j, p_i, t)\} \tag{3.3}$$

We assume that no SDC can access the global history. Each individual SDC maintains its own history to evaluate its partnership. For example, an SDC may prefer to lend its resource to the SDCs which gave it the maximum support in the past. In addition, given that resource sharing is free (in terms of money), each SDC in the decentralized cloud has an incentive to be selfish, e.g., borrowing more resources than lending. We define the *altruism level* as the ratio of resources borrowed to the amount lent. For the History-based strategy function, an SDC p_i can evaluate its altruism level with other SDC p_j based on its partner set:

$$al_i^j = \frac{r(p_j, p_i, t)}{r(p_i, p_j, t)}, p_j \in \Gamma(p_i), r \in H_i \tag{3.4}$$

An altruism level of 1.0 means that p_i has borrowed as much resources as it has lent. A selfish SDC, which has borrowed more than what it has lent, has an altruism level greater than 1 while a selfless SDC has an altruism level less than 1. It should be noted that each SDC evaluates its altruism level based on its private history, which is thus a local value rather than a global one. Based on the evaluation results, the SDCs can decide whether to lend the resource to their partners (for RSE), or to replace their partner with a new one (for PM).

3) Prediction-based: Another key determinant to lend resources to others is the arrival of future workload. The strategy function based on prediction will evaluate the impact of resource sharing in a probabilistic way since all three workload arrival models are based on a stochastic process. For this strategy, we have taken inspiration from predictive models for the arrival of workload in cloud which have been studied to allow elastic resource provisioning [4, 18]. At any given time t, after receiving a borrow request, an SDC p_i will evaluate the probability of incoming demand at time $t + 1$ greater than its available capacity, i.e., $P(d_i(t + 1) > c_i(t + 1))$, to decide if it should lend resources. Moreover, we define *risk indicator* as a threshold value to evaluate the risk of prediction. If the probability is greater than risk indicator, the SDC will not lend its resources and reject the requests from its partners. Different SDCs will set different values for risk indicator. For example, if an SDC is risk-averse, it will set indicator as low as possible, in order to minimize the risk of resource under-provisioning.

4) Reciprocity-based : In real cloud scenarios, one SDC usually will likely have a limited number of partners. This can happen due to geographical and legal constraints. The problem is to find out partners with high performance of cooperation. In our decentralized cloud, we initially build a partner set for each SDC merely based on the location. There is the possibility that the workload arrival patterns of two geographically neighboring SDCs are not compatible, e.g., both might reach their demand peaks at the same time, thus leading to poor performance. Since each of the SDCs has limited number of idle resources, it is ideal to develop a cooperative partnership such that the amount of resources lent to their partners can be exchanged for borrowing the same amount of resources. In our decentralized cloud scenario, the SDC with reciprocity-based strategy will replace the partners which reject its requests, with other new partners, which is similar to the Tit-for-Tat strategy in the BitTorrent protocol. It was in their book, *Prisoner's Dilemma*, that A. Rapoport and A. Chammah, first introduced the Tit-for-Tat

Strategy [93]. This fact often remains unmentioned in the literature. Later on it gained popularity through Axelord's tournament [85]. And finally, it inspired BitTorrent and other P2P systems [84, 94]. We define *tolerance* (τ) as the maximum number of rejections that an SDC can accept from its partners. When the number of rejections of one partner reaches the tolerance threshold, the SDC will replace this partner with a new one. Thus, at any time t, reciprocity-based strategy only depends on the history of last τ moments $H^{t-\tau}$, and thus it is unnecessary to keep all the history.

Name (Abbr.)	x Factors	RSE	PM
General (Gen)	$c_i(t), d_i(t), location$	$\surd$	
History (Hist)	$c_i(t), d_i(t), location, H$	$\surd$	$\surd$
Prediction (Pred)	$c_i(t), d_i(t), location, workload$	$\surd$	
Reciprocity (Recip)	$H^{t-\tau}, Location$		$\surd$

Table 3.1: Strategy function and its usage

All the above four strategies take location into account. In our decentralized cloud, we assume that each SDC prefers to collaborate with the nearest ones, which can improve the service response time. We conjecture that in practice this is how things will turn out since most SDCs would prefer not to cooperate with SDCs in distant geographical locations due to several issues including legal concerns and service response time. Table 3.1 summarizes four strategy functions and their main purpose. Furthermore, through combination of 3 RSE strategies and 2 PM strategies, we can have 6 more strategies. In summary, we will discuss the following 9 strategies in terms of three groups (the abbreviation of each strategy is listed in table 3.1): 1) *Strategies without PM*: Gen, Hist, Pred; 2) *Reciprocity-based strategies for PM*: Gen-Recip, Hist-Recip, Pred-Recip; 3) *History-based strategies for PM*: Gen-Hist, Pred-Hist, Hist-Hist.

3.5 Methodology

3.5.1 Simulator

We developed our simulator based on CloudSim [95] which is a simulation toolkit that enables modeling and simulation of cloud computing systems based on discrete events. The simulator consists of three modules, namely *DecentralizedCloud*, *WorkloadManager* and *CloudBroker*. The DecentralizedCloud module implements the networked SDCs, VM allocation and resource sharing protocols among the SDCs. WorkloadManager implements three different workloads and generates workload arrivals for cloud end-users. CloudBroker acts on behalf of cloud end-users, and is responsible for sending resource requests and submitting the workloads to the VMs. The simulator reads a configuration file based on XML format which contains specific values for the various X_i factors of the strategy functions, workload information and network topologies of the SDCs.

3.5.2 Experimental Setup

To simulate the decentralized cloud, we set up the basic configurations for the workload arrival models, workload execution time and the SDCs.

Workload Arrival Models: By default, we use the same configuration as discussed in section 3.3.2.

Workload Execution Time: To model the variability in workload execution time, we assume the execution time is distributed as follows: when a workload is generated with probability 0.8, the execution

time is uniformly distributed in [0.5, 10] time units; with probability of 0.18, it is uniformly distributed in the interval [10, 50] and finally with probability of 0.02, it is uniformly distributed from 90 to 100 time units. Thus, the average execution time is about 11.5 and the maximum execution time is 100.

Small Data Center: All the SDCs have homogeneous physical server machines (with 6 cores, 24GB memory and 1TB disk space) and each of them can host three VMs (with 2 cores, 8GB memory and 300GB disk space). The total capacity of an SDC is the number of server machines which is uniformly distributed in [370,420]. The distance between two neighboring SDCs is also uniformly distributed from 10 kilometers to 30 kilometers. We note that the network latency between two SDCs depends on the distance since we ignore the cost of data transfer.

3.5.3 Metrics

We use the following metrics to evaluate the performance of the decentralized cloud:

Request Loss is the accumulated amount of under-provisioned resources in the decentralized cloud, i.e., $\sum_{i=1}^{n} loss_i$.

Resource Over-provisioning is the accumulated amount of over-provisioned resources in the decentralized cloud, which is computed as the total number of the hours when the VMs in the decentralized cloud are in the idle status.

Service Response Time describes the performance of the decentralized cloud from the perspective of cloud end-users and evaluates how long the cloud-user must wait for a response to a resource request.

3.6 Results and Discussion

3.6.1 Incentive Analysis

3.6.1.1 Resource provisioning dilemma

We firstly simulate a single SDC to find out the impact of different workloads on resource provisioning. We vary the total capacity of an SDC with three different workloads which are shown in Figure 3.1. It can be clearly seen in Figure 3.2 that the increase in the total capacity of an SDC leads to a slow decline in the amount of request losses (Figure 3.2 (a)) but fast growth in the accumulated amount of idle resources (Figure 3.2 (b)). From the perspective of resource over-provisioning, we can see all three workload models show similar performance. It is because all three workload arrivals have the same mean arrival rate and therefore the total amount of workloads throughout the simulation is almost the same. The difference between the amount of idle resources under different workload arrivals is caused by the different levels of request burstiness of the three workload arrivals. On the other hand, we can observe in Figure 3.2 (a) from the perspective of request loss, resource provisioning of an SDC with heavy-tailed arrival also presents a heavy-tailed characteristic. It can be also observed that if an SDC wants to achieve zero-loss of requests, the SDC with heavy-tailed arrival has to increase its capacity to at least 650 while in case of the Poisson arrival, an SDC only needs 450. This implies that in order to accommodate the peak of workload arrival, the SDC with heavy-tailed arrival has to over-provision nearly 0.5 times more than that of the Poisson arrival. Thus, in the real cloud scenarios, the SDCs with large variance in their workload arrivals have to confront the dilemma of resource provisioning. They should consider the tradeoffs between the request loss and the cost of resource over-provisioning. For example, the SDC with heavy-tailed arrival in Figure 3.2 (a) may only provision for 500 capacity and

incur the loss of about 150 resource requests, while saving the cost of maintaining an additional 150 more server machines required to achieve zero loss (at 650 capacity in the Figure 3.2 (a)).

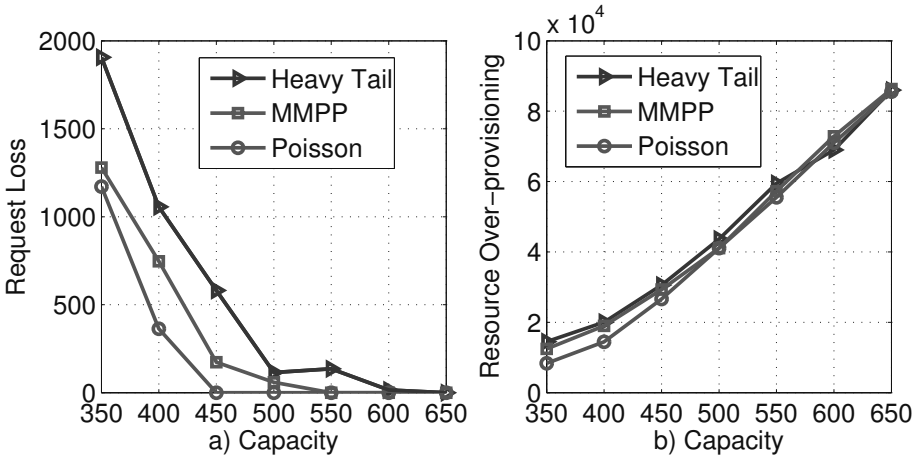

Figure 3.2: Resource provisioning with different capacity

Figure 3.2 can also help us in analyzing the incentives of cooperation in the decentralized cloud. On one hand, there are surplus resources, which is the necessary condition of enabling resource sharing. We can see this in Figure 3.2 (b), that no matter how the capacity and workload model change, the SDCs on the whole always have a considerable amount of idle resources. On the other hand, there is a great demand of resource sharing to avoid both the loss of resource requests and the monetary cost of buying more resources. Resource sharing in the decentralized cloud provides a fast way to provision additional resources, thereby avoiding the delay in meeting user demand and revenue generation. In fact, in the real world cloud scenario, we believe there are more incentives for cooperation other other revenue growth. For example, finding the resources in specific location improves service response time and/or suitably addresses legal and privacy issues.

3.6.1.2 The Impact of Cooperation

From Figure 3.2 (a), we can conclude that with capacities in the range [370,420], all the SDCs will suffer request losses if they run individually. In order to avoid these losses, the SDCs can cooperate with each other. To further analyze the impact of cooperation, we do another set of experiments. We simulate a decentralized cloud with 25 SDCs in three different modes to evaluate the impact of cooperation among the SDCs. We randomly assign the workload arrival model and capacity to each SDC. We assume the utilization of each SDC is uniformly distributed in the interval [0.8, 0.9]. With the mean workload arrival rate $\lambda = 100$ and the average workload execution time, we generate the capacities of 25 SDCs which are uniformly distributed from 370 to 420.

Firstly, we simulate the decentralized cloud in a non-cooperative way which means no cooperation exists among the SDCs ($|\Gamma(p)| = \{\emptyset\}$). This will serve as the *worst case scenario*. In Figure 3.3 we can observe that in the non-cooperative case, the total request loss is as high as about 15,000. Now, we introduce cooperation to the decentralized cloud where all of the SDCs adopt the general strategy to decide whether they share the resource or not. If the decentralized cloud is fully cooperative, i.e., all the

SDCs can share resources with each other, the loss will decrease drastically to about 1200, only 8% of the request loss in the non-cooperative mode as can be seen in the Figure. This will serve as the *best case scenario*. We argue that in the real cloud scenario, cooperating SDCs would not be cooperating with a great number of partners due to geographical and legal constraints, among other factors. Therefore, the performance of cooperation between SDCs would be somewhere between the worst-case and best-case scenarios. As a validation of the above point, it can be seen from Figure 3.3 that with each SDC having 2 resource sharing partners, the request loss of the decentralized cloud will decrease to about 10,000. Moreover, the loss will fall by nearly 40% (of that with 2 partners) if the number of partners increases to 5, i.e., $|\Gamma(p)| = 5$. Both these values are between the worst case and best case scenarios.

Therefore, from the Figure 3.3, we can find that the cooperation among the SDCs in the decentralized cloud greatly reduces the loss of resource requests. In addition, the more partners an SDC has, the more the resource requests are served in the decentralized cloud. However, there are new problems that emerge. Since different SDCs have different workload arrival patterns, capacities and strategies for resource allocation, the following issues need to be addressed: How can we determine the performance of different strategies? How to form partnerships based on good sharing behaviors and shared interests and needs? Moreover, does there exist a strategy that outperforms others in a heterogeneous environment, and thus is robust? In the following, we will try to answer these questions with extensive evaluations.

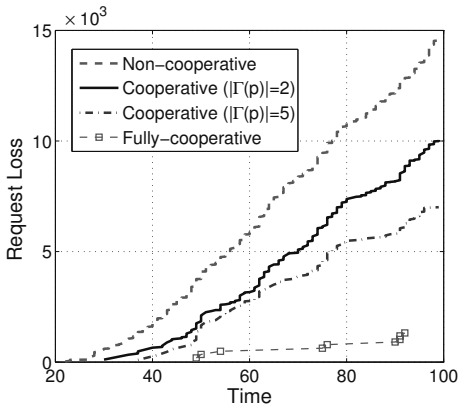

Figure 3.3: Impact of cooperation

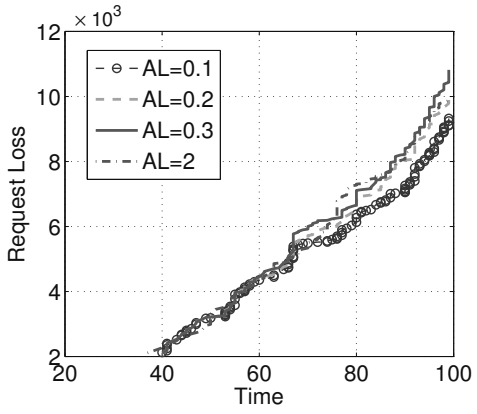

Figure 3.4: Varying altruism level

3.6.2 The Impact of Strategies

In this section, we will evaluate the three groups of strategies described in Section 3.4.

3.6.2.1 History-based strategy

An SDC with history-based strategy will evaluate the performance of cooperation based on the past interaction history. Specifically, every SDC will save the borrowing/lending interactions for future decision making. When the SDC (p_i) receives borrow request from the other SDC ($p_j \in \Gamma(p_i)$), it will firstly compute its altruism level (al_i^j) with p_j , which is the ratio of total amount of resources borrowed from p_j to the amount of resources lent to p_j. If $al_i^j > 1$, it means p_i has borrowed more resources than it has lent to p_j. Therefore, the higher the altruism level is, the more selfish p_i is. After computing the al_i^j,

it will compare the value with a threshold altruism level. Only if the altruism level is greater than the threshold, p_i will lend its resource to p_j. The effects of varying this threshold value are shown in Figure 3.4. It is interesting to observe that the smaller the threshold is, the better the performance is. Put another way, this strategy encourages the SDCs to be selfless so that they can suffer less request loss.

3.6.2.2 Prediction-based

Prediction-based strategy enables an SDC to predict future demand. Suppose an SDC p_i receives a borrow request of m resources, it will firstly compute the moving average of workload finishing rate $f_i(t)$ based on history. Then the predicted free capacity of p_i at the time $t+1$ is $c_i(t+1) = c_i(t) + f_i(t) - m$. Then p_i will compute the probability that the next demand $d_i(t+1)$ is greater than the future free capacity, i.e. $P(d_i(t+1) > c_i(t+1))$, based on cumulative distribution function. The probability approximating to 1 means that the risk that p_i cannot serve the demand at time $t+1$ is high. Therefore, if the SDC is risk-averse, it will set a risk indicator as low as possible to minimize the risk. We vary the risk indicator in the interval (0,1) and the results are shown in the Figure 3.5. We can see that the SDCs with the risk indicator 0.6 have the minimal loss while those with the risk either too small (0.2) or too large (0.8) will suffer more request loss. Thus we discover a *tradeoff* between being risk-averse and risk-seeking. On the one hand, if all the SDCs are risk-averse, they will be reluctant to cooperate so that the prediction-based strategy will reduce to a non-cooperative model. On the other hand, if all of them are risk-seeking, they will accept the borrowing requests even when the risk is high, due to which the performance reduces to that of the general strategy. Therefore, we argue that it is important to have moderate risk estimation to prevent future loss of requests.

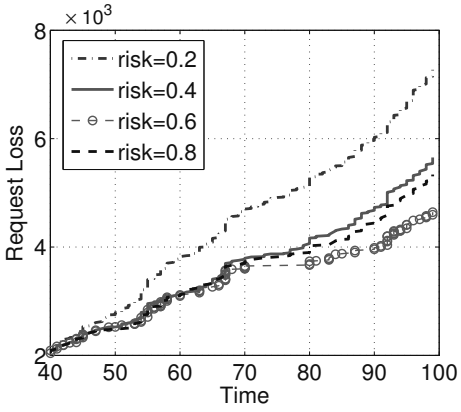

Figure 3.5: Varying risk indicator

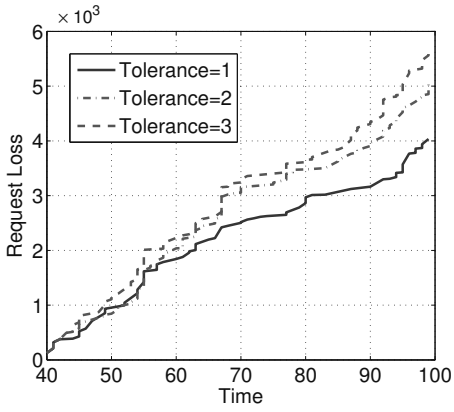

Figure 3.6: Varying tolerance

3.6.2.3 Reciprocity-based

Reciprocity-based strategy is employed to manage the partner set. At the beginning, we assume each SDC finds its partners based on the location proximity. The maximum number of partners is 5 for each SDC in the decentralized cloud. The SDCs will select new partners based on location if they want to replace the less cooperative partners. However, if no new partners are found, the SDCs will simply remove the less cooperative partners from the set. Meanwhile, each SDC maintains a block list which

helps it avoid the selection of removed partners in the future. To improve the forgiveness of the strategy, the block list will be emptied at regular intervals. We vary the tolerance from 1 to 3, respectively. It can be clearly seen in Figure 3.6 that strategy with the tolerance equaling 1 has the best performance in avoiding request loss. The other two strategies are more forgiving versions in that they do not punish isolated rejections. However, the results show that an excess of forgiveness is costly. The precise level of forgiveness that is optimal depends upon the environment. Though, results show that in our decentralized cloud setting, it is better for the SDCs to set the tolerance as 1 for partnership management.

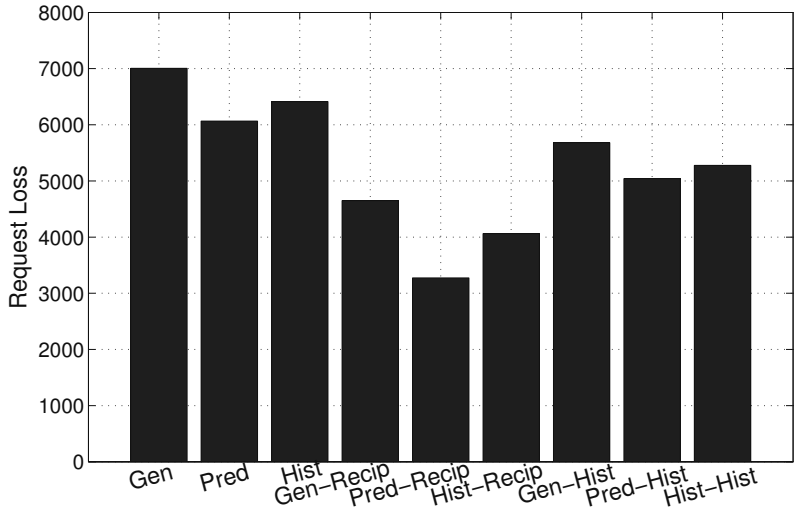

Figure 3.7: Performance of combined strategies

3.6.2.4 Strategy combination

Through the evaluation of each strategy, we derive the near-optimal parameter values for each strategy. In the following simulation, we will set the altruism level as 0.1 for history-based strategy, risk indicator as 0.6 for prediction-based strategy and the tolerance as 1 for reciprocity-based strategy. We apply those strategies into RSE and PM with combination and the results are shown the Figure 3.7.

In this set of experiments, we will vary the strategy of the SDCs while keeping other parameters such as workload and capacity unchanged. We analyze the impact on request loss and service response time. From Figure 3.7, we can safely reach two conclusions: 1) the strategies with partnership management improve the performance greatly, which means that it is important for the SDCs to find good partners. 2) The various strategies based on reciprocity outperform the others. This property enables the SDCs to manage their partnerships effectively only relying on the last decisions of their partners, rather than the large amount of past interaction histories. Besides that, we also notice that prediction-based strategies for RSE are better than the other two RSE strategies. The reason is that, in decentralized cloud model, we have deterministic workload arrival models. However, in the real cloud scenarios, it is difficult to predict the future demand and therefore the prediction-based strategy may not outperform the other two. Nevertheless, we can still apply the reciprocity-based strategy to achieve better performance.

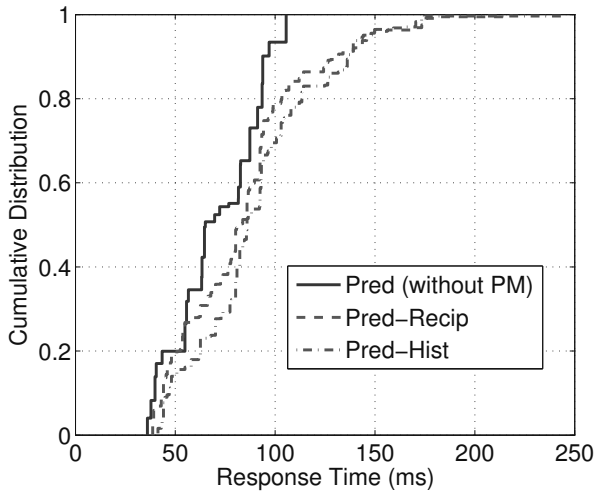

Figure 3.8: Impact of Partnership Management (PM) on response time

On the other hand, with introduction of partnership management, the new selected partners may be located far away, which exerts negative influence on the service response time. If we view each discrete time unit as 1 hour, the network latency between two neighboring SDCs is uniformly distributed in [10ms,30ms] which is determined by the distance. In Figure 3.8, we explore the impact of different PM strategies on the service response time. It shows three strategies with the same Prediction-based strategy for RSE but different strategies for PM. We can observe that compared to the strategy without PM in which only about 8% requests have response time larger than 100ms, the response time of the strategies with PM increases remarkably with about 20%-25% of requests having response time larger than 100ms. In addition, we can observe the heavy-tailed distribution in response time of the strategies with the PM, where a proportion of the response time (about 1%) can be as high as 250ms. Thus, SDCs have to find the tradeoffs between the service response time and the request loss. For example, if an SDC wants to guarantee the service response time for its users, it shall consider the maximum distance that its partners can be located. To put it bluntly, it is not advisable for an SDC in Europe to cooperate with an SDC in Japan (the assumption here being that an SDC and its users would be in the same vicinity).

3.6.2.5 Individual performance evaluation

In the previous discussion, we view the decentralized cloud as a society and evaluate the performance from the perspective of social welfare. In this subsection, we evaluate the performance from an individual SDC's point of view. We randomly assign the workload arrival models for each SDC. There are 7 SDCs (1,3,4,7,11,17,22) with the Poisson arrival, 9 SDCs (2,6,8,9,10,13,14,21,24,25) with MMPP and 8 SDCs (5,12,15,16,18,19,20,23) with heavy-tailed arrival. Figure 3.9 shows the performance of each individual SDC under different strategies. The color of each grid, from black to white, in Figure 3.9 represents the request loss of an SDC i.e., more black means less loss. We can either compare the performance of different strategies by column or compare the performance of different SDCs by row. From the results, several interesting observations can be made: 1) the first column represents the performance in non-cooperative environment, which has the worst performance due to no resource sharing. The columns

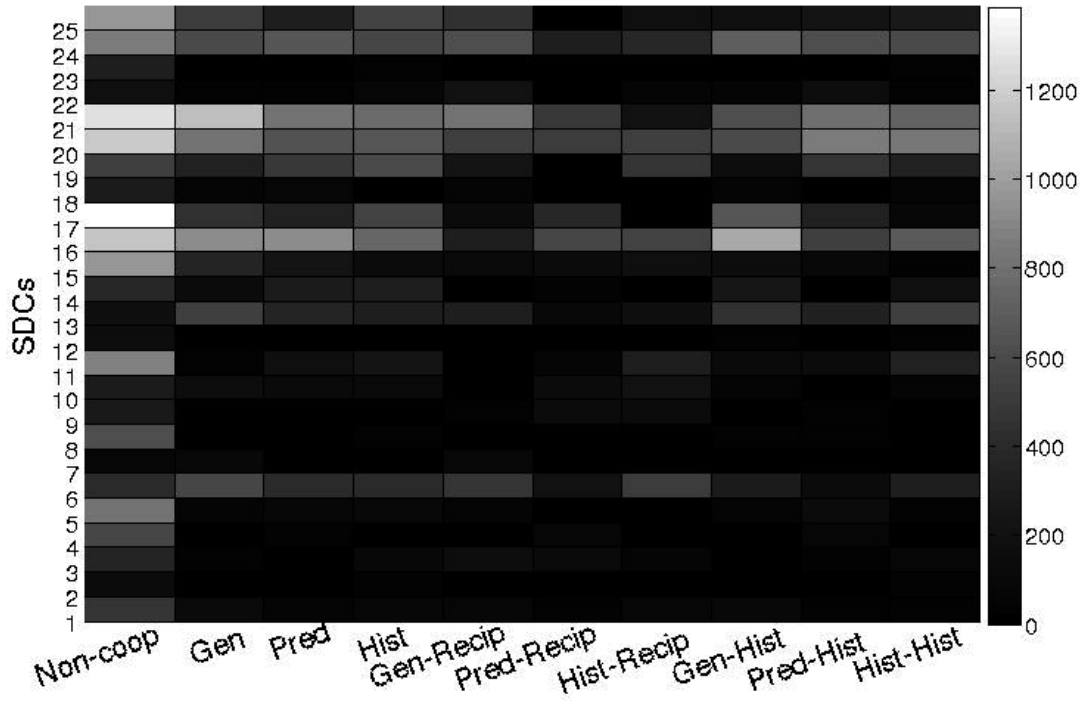

Figure 3.9: Individual SDC performance

with reciprocity-based strategies are better than other columns since the color is almost entirely covered by black. Especially the column with Pred-Recip strategy performs generally well no matter what the workload arrival. 2) With the adoption of one appropriate strategy, any SDC in the decentralized cloud can improve their performance. In the Figure 3.9, we can always find a strategy for an SDC to suffer less loss than that in the first column which is the worst performance due to non-cooperation. For example, the best choice for SDC 21 is Hist-Recip while that for SDC 16 is Gen-Recip. This means it is necessary for the SDCs to learn that how to discover the appropriate strategy under different conditions. 3) If we use the difference between the best and the worst performance to evaluate the incentives of the SDCs, we find out the incentives of different SDCs for cooperation in decentralized cloud vary a lot. For example, for SDCs 2 or 7, there is no striking improvement for them to participate in the decentralized cloud. Since they find little incentive to cooperate, they may leave the decentralized cloud. While for SDC 17 and 21, there is a strong incentive to cooperate with others since their request loss can reduce dramatically through cooperation. If we take a further step to correlate incentive with other features of the SDC, we find that the variance in incentives has weak correlation with the SDC's capacity or workload type. This is because we observe that most of the SDCs with weak incentives have dissimilar workload type or capacity. Given that both the arrival rate and the workload execution time also have the same average, we infer that there is no predominant factor to determine the incentives of the SDCs to participate in the decentralized cloud. Therefore, there will be many SDCs, *regardless of capacity or workload type*, which will join the decentralized cloud and achieve better performance through cooperation.

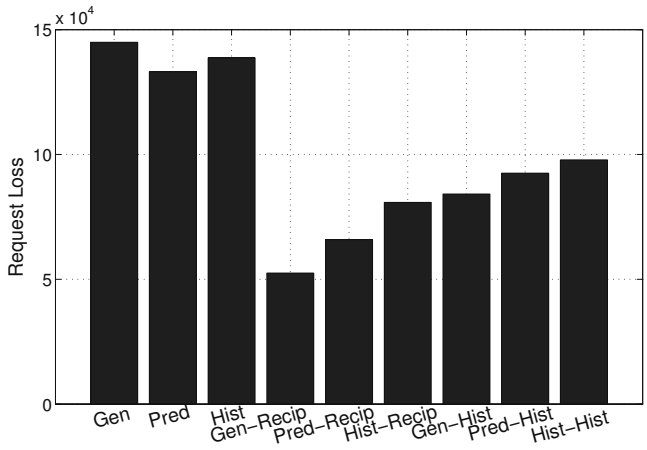

Figure 3.10: Performance with a simulation time of 1000

3.6.2.6 Long-scale simulation

In this part, we extend the simulation to a duration of 1000 to observe long term performance of different strategies. If 1 time unit is assumed to be 1 hour, the duration of simulation is about 42 days. The results are shown in Figure 3.10. Again, we observe that reciprocity-based strategies outperform the others. Meanwhile, it is interesting to see that the Gen-Recip is the best one rather than Pred-Recip as in Figure 3.7. We infer that in the long run, the performance of prediction-based strategy largely depends on the prediction precision. As discussed earlier, in the real cloud scenario, it could be very difficult to predict demand and therefore the losses incurred through misprediction will accumulate as time goes by. And this is exactly what our results show.

However, it is worthwhile to note that Gen-Recip based strategy has a good performance, which almost reduces the request loss by over 55% as compared to the General strategy. Moreover, it gives us two important implications: 1) We observe that the Gen-Recip based strategy is better than those based on history and prediction. That means it is better for SDCs to be generous to their partners rather than to judge them based on history or prediction. We believe that in the real world cloud scenarios, most of the SDCs cannot be fully rational since they do not have complete information to perform a perfect evaluation of their partners' requests based on history or prediction. It is better to keep it simple, and to offer help to their partners as long as they can. 2) We find out that as time passes, the optimal strategy for SDCs will change. In a decentralized cloud with high heterogeneity, the optimal strategy for each SDC may vary. To derive the optimal strategy, the SDCs have to employ an ongoing process of learning.

3.6.3 Competition of Strategies in a Heterogeneous Setting

In previous discussion, all the SDCs in the decentralized cloud adopt the same strategies. In this subsection, we will simulate a decentralized cloud of 25 SDCs with heterogeneous strategies. The goal is to discover which strategy can thrive in a heterogeneous environment. In this set of experiments, we exclude three strategies without PM and only consider six strategies based on reciprocity and history with PM. The strategy of each SDC is assigned randomly but the parameters of each type of strategy are the same, with risk indicator as 0.6, altruism level as 0.1 and tolerance as 1.

In this simulation, we design an adaptive strategy selection algorithm which is similar to the selection in genetic algorithms. However, we do not do explicit population wide selection, since in our case different SDCs can have different resources (as in capacity) and different needs (location and privacy concerns). Therefore, each SDC is interested in the strategy that works best for it. Our selection algorithm has two phases: training and testing. The time length of two phases are the same. Moreover, for each SDC, we define a *fitness function* to evaluate the performance of each strategy. Throughout the training phase, each SDC will change its strategy to a new random strategy at regular intervals. In other words, if we set the length of training phase to be 300 time units, the expected training time of each strategy is 50 time units in each training phase. At the end of training phase, the fitness function will compare the request loss of each strategy and return a strategy with the minimal request loss. After that the SDC will use this selected strategy without changing for the entire testing phase. The SDC will compute the total loss in the both training $loss_{train}$ and testing phase $loss_{test}$. It is clear that if the selected strategy under testing is the optimal one, the loss in the testing phrase must be less than that in the training phrases. Thus, in the end of the testing phase, the SDC will compare the loss of two phases. If $loss_{test}$ is greater than $loss_{train}$, it will switch to training phase to find other potential better strategies. Otherwise if $loss_{test}$ is less than $loss_{train}$, it will proceed with the next testing phase to continuously verify the selected strategy. This algorithm simulates the dynamics in the decentralized cloud in a random way and learns the performance of each strategy in a trial-and-error manner. This process will iterate until the end of simulation.

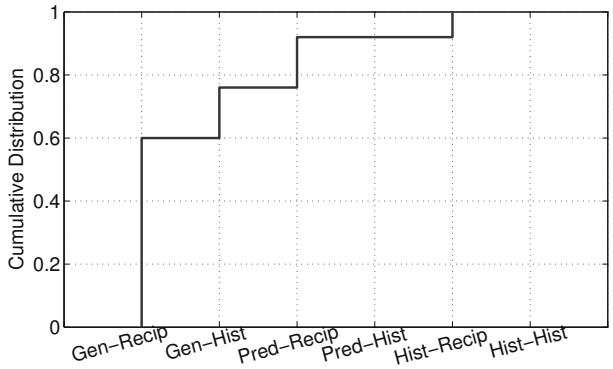

Figure 3.11: Competition of strategies

We set the length of both training and testing phases as 200. During the training phase, we change the strategy every 10 time units, and the entire duration of simulation is 1800 time units. Figure 3.11 shows that 60% of all 25 SDCs choose the Gen-Recip strategy as their best choice while the total number of Reciprocity-based strategies takes up 84% with only 4 SDCs adopting Gen-Hist strategy. It is interesting to observe that the experiment also simulates the survival of the fittest, which both the Pred-Hist and Hist-Hist strategies cannot survive during the competition since no SDC adopts those two strategies. We also observe that three of the four SDCs that choose the Gen-Hist strategy have heavy-tailed arrivals. We infer that history plays a more positive role in finding better partners for the SDCs with high burstiness (as in the case of heavy tail arrivals). Combining those results with the ones in Figure 3.10, it is safe to conclude that, in the long run the reciprocity-based strategy is the most effective strategy under varying

conditions. We also increase the interval of changing a new strategy to 20 and 30 time units to observe the effect of longer running times on the fitness of the strategies, and observe similar results.

3.6.4 Discussion

Based on our extensive simulation of realistic decentralized cloud, we can answer the three questions raised in the beginning. To overcome the resource provisioning dilemma through cooperation in the decentralized cloud, different SDCs can employ a wide range of more or less sophisticated strategies. We observe that the performance of history-based strategies depends on the altruism level while that of prediction-based strategies on the risk indicator. For history based strategies, a strategy with high performance is the one that encourages selfless behavior, while for the prediction based strategies, the one with moderate risk evaluation is successful. Also, we discovered that the reciprocity-based strategies are the simplest yet the most effective way to develop cooperation in a variegated decentralized cloud. Compared to other strategies, the reciprocity-based strategies can immediately produce both cooperation and retaliation for their partners, rather than relying on history or prediction. This is a counter-intuitive result since on first thought, it would appear that prediction analysis applied on historical data would allow for better performance.

Furthermore, in our competition setting, we observe that the reciprocity-based strategies can also thrive in a dynamic environment with different strategies. The competition models the learning process of an SDC, through trial and error, to reach the optimal strategy. Through many trials, the SDCs can find the fittest strategy eventually. However, this process of learning might take a long time to move slowly toward mutually rewarding strategies. In the real world, the cost of learning for SDCs is high and they may not have enough patience to try. Thus, our results pave the way for the SDCs to speed up the learning process.

Nevertheless, our model is not without its limitations. First, validating our model with realistic workload dataset rather than synthetic ones would give us more confidence in our results. Second, even though we consider some intuitive strategies, the design space of the strategies can be huge (e.g., strategies that capture time of day correlations *or* distribution of request demand across multiple partners instead of rejecting the request if one partner cannot satisfy it fully, as in our current model). Thus we cannot be sure of the goodness of our 'winning' strategies in face of unknown strategy variants. Finally, in our model, we mostly focus on the request loss of the SDCs while a more general model shall incorporate other factors related to the QoS.

3.7 Conclusion and Future Work

This chapter has proposed a decentralized cloud model in which a group of networked SDCs can work collaboratively to overcome the limitations raised by massive centralized cloud infrastructure. We also present a general strategy function to evaluate the performance of cooperation based on different dimensions of resource sharing. Our results show that the reciprocity-based strategies are more effective than other strategies, which can help the SDCs improve the performance of cooperation. In the future, we are interested in comparing our model with the one based on monetary payment. Furthermore, we also plan on evaluating our approach using real workload datasets.

Part III

Optimization

Chapter 4

Optimizing Information Leakage in Mutlicloud Storage System

The final value of life lies in the awakening and the ability to think, not only is survival.

Aristotle

4.1 Introduction

4.1.1 Motivation and Challenges

With the increasingly rapid uptake of devices such as laptops, cellphones and tablets, users require an ubiquitous and massive network storage to handle their ever-growing digital lives. To meet these demands, many cloud-based storage and file sharing services such as Dropbox, Google Drive and Amazon S3, have gained popularity due to the easy-to-use interface and low storage cost. However, these centralized cloud storage services are criticized for grabbing the control of users' data, which allows storage providers to run analytics for marketing and advertising [1]. Also, the information in users' data can be leaked e.g., by means of malicious insiders, backdoors, bribe and coercion. One possible solution to reduce the risk of information leakage is to employ multicloud storage systems [3, 22, 30, 31] in which no single point of attack can leak all the information. A malicious entity, such as the one revealed in recent attacks on privacy [96], would be required to coerce all the different CSPs on which a user might place her data, in order to get a complete picture of her data. Put simply, as the saying goes, do not put all the eggs in one basket.

Yet, the situation is not so simple. CSPs such as Dropbox, among many others, employ *rsync*-like protocols [97] to synchronize the local file to remote file in their centralized clouds [98]. Every local file is partitioned into small chunks and these chunks are hashed with fingerprinting algorithms such as *SHA-1, MD5*. Thus, a file's contents can be uniquely identified by this list of hashes. For each update of local file, only chunks with changed hashes will be uploaded to the cloud. This synchronization based on hashes is different from *diff*-like protocols that are based on comparing two versions of the same file line by line and can detect the exact updates and only upload these updates in a patch style [97]. Instead,

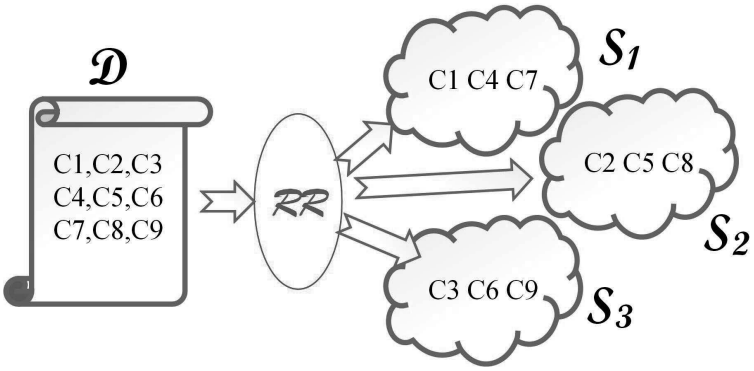

Figure 4.1: The motivating example

the hash-based synchronization model needs to upload the whole chunks with changed hashes to the cloud. Thus, in the multicloud environment, two chunks differing only very slightly can be distributed to two different clouds. The following motivating example will show that if chunks of a user's data are assigned to different CSPs in an unplanned manner, the information leaked to each CSP can be higher than expected. Suppose that we have a storage service with three CSPs S_1, S_2, S_3 and a user's dataset D. All the user's data will be firstly chunked and then uploaded to different clouds. The dataset D is represented as a set of hashes generated by each data chunk. This scenario is shown in Figure 4.1. In addition, we consider that the data chunks are distributed to different clouds in a round robin (RR) way. Apparently, RR is good for balancing the storage load and each cloud thus obtains the same amount of data. However, *the same amount of data does not necessarily mean the same amount of information.* For example, if we find that the set of chunks $\{C3, C6, C9\}$ are almost same, it means S_3 actually obtains the information equivalent to that in only one chunk. If all other chunks are different, S_1 and S_2 obtain three times as much information as S_3, even though all of them obtain the same amount of data. The problem does not exist in a single storage cloud such as Dropbox since users have no other choice but to give all their information to only one cloud. When the storage is in the multicloud, we have the opportunity to minimize the total information that is leaked to each CSP. The optimal case is that each CSP obtains the same amount of information. In our example, data distribution based on RR can achieve the optimal result only if all the chunks are different. However this is not the case in cloud storage service due to two reasons: 1) Frequent modifications of files by users result in large amount of similar chunks[1]; and 2) Similar chunks across files, due to which existing CSPs use the data deduplication technique.

In fact, the data deduplication technique, which is widely adopted by current cloud storage services like Dropbox, is one example of exploiting the similarities among different data chunks to save disk space and avoid data retransmission [98], [99]. It identifies the same data chunks by their fingerprints which are generated by fingerprinting algorithms such as *SHA-1, MD5*. Any change to the data will produce a very different fingerprint with high probability [100]. However, these fingerprints can only detect whether or not the data nodes are duplicate, which is only good for *exact equality* testing. Determining identical chunks is relatively straightforward but efficiently determining similarity between chunks is an intricate task due to the lack of similarity preserving fingerprints (or signatures). At the same time, similarity is of paramount importance if one wants to limit information disclosure. Put simply, two paragraphs of text

[1]Most CSPs maintain revision history.

with one word different would lead to two different chunks. If one were to only consider identity, the two chunks would be considered different and placed separately; however both of them contain almost entirely the same information, hence they should ideally be placed together. We note here that the above problem is relevant even with encryption because once the encryption key is exposed (as in coercion of the CSP by some third party such as the National Security Agency or due to the maliciousness of the CSP itself), the entire data of the user can be easily leaked. If encryption is performed after detecting near duplicate chunks and placing them together, then the information leakage can be reduced even if the encryption key is exposed. Therefore, we need more sophisticated techniques to detect the near-duplicate (or similar) data chunks to reduce the information leakage in the multicloud storage system.

4.1.2 Approach and Contributions

Through the above example, we can see that storing the data in a multicloud system without proper optimization on the data distribution can lead to avoidable information leakage. In this chapter, we focus on reducing information leakage to each individual CSP in a multicloud storage system and provide mechanisms for distributing users data over multiple CSPs in a leakage aware manner. First we provide a novel algorithm for generating similarity preserving signatures for data chunks. Next based on this algorithm, we devise a chunk placement storage plan that efficiently synchronizes similar chunks together in a multicloud environment. Finally, we evaluate and validate our design using real datasets. Specifically, we make the following contributions in this chapter:

- We present *StoreSim*, an information leakage aware multicloud storage system which incorporates three important distributed entities and we also formulate information leakage optimization problem in multicloud.
- We propose an approximate algorithm, *BFSMinHash*, based on Minhash and Bloom filter to generate similarity-preserving signatures for data chunks. We also design a pairwise information leakage function based on Jaccard similarity.
- Based on the information leakage measured by BFSMinHash, we develop an efficient storage plan generation algorithm, *SPClustering*, for distributing users data to different clouds.
- Finally, we use two datasets crawled from *Wikipedia* and *GitHub*, containing files with multiple revisions, to evaluate our framework. Through extensive experiments, we show the effectiveness and efficiency of our proposed scheme for reducing information leakage across multiple clouds. Furthermore, our analysis on the system attackability demonstrates that StoreSim makes attacks on information much more complex.

The rest of this chapter is organized as follows. In section 4.2, we introduce multicloud storage services and discuss data synchronization mechanisms among three interacting entities in the multicloud. In section 4.3, we present the architecture, models and storage protocols of our StoreSim. Section 4.4 details the BFSMinHash algorithm to generate similarity-preserving signatures for data chunk while section 4.5 presents the SPClustering algorithm to distribute users data to different clouds with respect to optimizing information leakage. In section 4.6, we discuss our experiments results and also the limitations of StoreSim. In section 4.7, we review existing literature related to our work and explain why our work is different. Finally, section 4.8 concludes our work.

4.2 Multicloud Storage Services

In this section, we first introduce multicloud storage services from the perspectives of both distribution and optimization. Then we discuss data synchronization mechanisms among three distributed entities in multicloud storage services.

4.2.1 Distribution and Optimization

Cloud storage services such as Dropbox and Google Drive, in essence, are centralized repositories for vast aggregations of personal data which can be monetized to afford the low-cost (or free) storage services for their users. While the users enjoy these storage services, they also lose their control on the data. Recent news about PRISM [96] shows that these CSPs can be compromised under coercion. Some other cloud storage services such as Wuala, SpiderOak employ client-side encryption to encrypt all the data before uploading the data. However, this does not change the inherent nature of centralized architecture. As discussed previously, even with encryption, once the encryption key is exposed (e.g., by leveraging backdoors in the key-generation software [101], by compromising the insiders who know the key or by compromising the devices that stores the key), a user's entire data can be easily divulged. The situation can be somewhat alleviated by using multiple clouds services so that no single CSP has access to the user's entire data (encrypted or otherwise). Many works have been proposed in both academia [3, 22, 30, 31] and industry [32, 33, 34] for using multiple CSPs for storing data. These works show that data *distribution* over multiple CSPs can avoid single point of failure, thereby improving the service availability and fault-tolerance. In addition, adopting multiple CSPs offers the opportunities for *optimization* on different metrics such as cost, network latency, service response time and vendor lock-in. Unlike the previous work, we focus on the optimization of information leakage in multiple storage services (against single point of attack).

4.2.2 Data Synchronization Mechanism of Cloud Storage Services

In multicloud storage system, there are three distributed entities which synchronize users' data from the remote client to the cloud:

1) **Client** is in charge of pre-processing the users' data for the purpose of optimization, such as chunking (i.e., dividing files into individual chunks of a maximum size data unit), deduplication (i.e., avoiding storing and re-transmitting the same content already available on the remote servers), delta encoding (i.e., transmission of only modified portions of a file), bundling (i.e., the transmission of multiple small files as a single object) and encryption/decryption;

2) **Metadata servers** are used to store the metadata database about the information of files, CSPs and users, which usually are structured data representing the whole cloud file system;

3) **Storage servers** store the raw data blocks which can be both structured and unstructured data.

The most essential step of data synchronization is to detect updates. One solution is *diff-like* protocols [102] which are based on comparing two versions of the same file line by line and can detect the exact updates. Only these updates will be uploaded to the cloud in a patch file which describes the difference between the old and the new version. However, *diff-like* protocols are not suitable for cloud storage services for three reasons. First, to compute the patch file, the client needs more storage overhead to store old versions, leading to the loss of users. Second, cloud storage services usually synchronize users' files

across different clients and devices. If a file is modified in one client, then all other clients need to update both the old and the new version of this file, which results in high communication overhead. Last but not least, cloud storage services will be in great danger if the client bears the burden of maintaining revision histories. For example, a mistake of deleting old versions made by users can result in synchronization errors.

Instead of using *diff-like* protocols, CSPs such as Dropbox, among many others, employ *rsync*-like protocols [97] to synchronize the local file to remote file in their centralized clouds [98]. *rsync*-like protocols only require each client only storing the newest version and use signature-based approach to detect updates. Specifically, every local file in the client is partitioned into small chunks and these chunks are hashed with fingerprinting algorithms such as *SHA-1, MD5*. In this way, a file's contents can be uniquely identified by this list of hashes and we call these hashes as signatures. To synchronize the updates to the cloud, the client will firstly send the signatures of current file to the metadata server. Then, the metadata server will detect these modified chunks by comparing current signatures with the signatures of last version and only returns the signatures of these changed chunks to the client. Finally, the client will only upload these chunks with changed signatures to the storage server. In this chapter, we design our system based on *rsync*-like protocols and further optimize the system in terms of information leakage.

In the next section, we present *StoreSim*, an information leakage aware system for multicloud storage service with considering three distributed entities and their interactions.

4.3 StoreSim

In this section, we firstly describe the architecture of *StoreSim*. Then we introduce StoreSim in terms of metadata and CSP models. Finally, we formulate the information leakage optimization problem in the multicloud.

4.3.1 Architecture

The architecture of StoreSim is shown in Figure 4.2. It can be observed that there is a trust boundary between the metadata and storage servers. We assume that clients and metadata servers, which are situated inside the trust boundary, are trustable by users while remote servers outside the boundary are untrustworthy. For example, the metadata can be stored in private database servers while storage servers can be located in public CSPs such as Amazon S3, Dropbox and Google Drive. Storage servers can be accessed through standard APIs (Application Programming Interfaces). As is shown in Figure 4.2, all control flows are inside the trust boundary while data flows can cross the trust boundary. In order to optimize the information leakage, we design two components in *StoreSim*. The first component is the Leakage Measure layer (LMLayer) that is used to evaluate the information leakage and further to generate the storage plan which maps data chunks to different clouds. The other component is the Cloud Manager layer (CMLayer) that provides cloud interoperability in a syntactic way. In the following, we will first present how we model metadata and storage cloud.

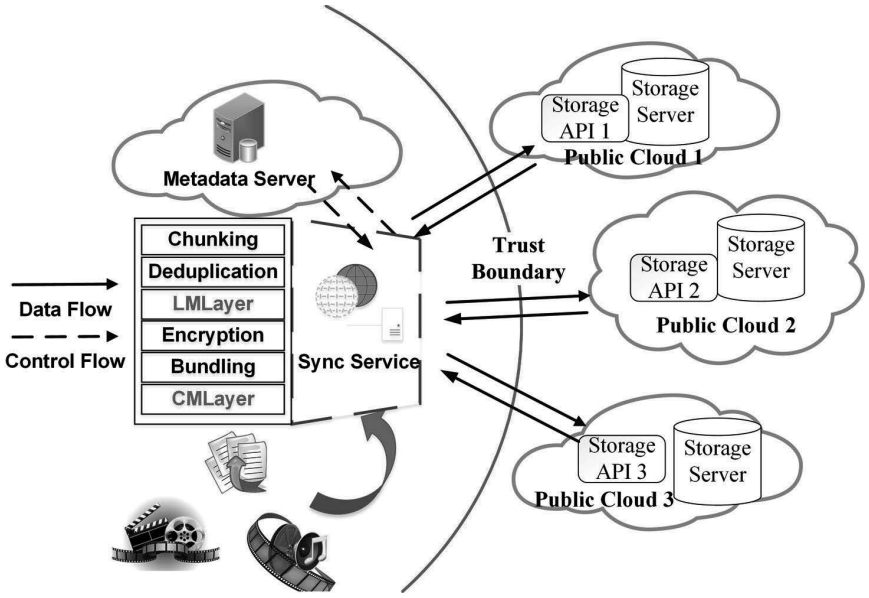

Figure 4.2: Architecture of StoreSim

4.3.2 MetaData Model

The data model we discuss in this section is for the metadata that represents the file system of StoreSim. We model users' data as a labeled graph $\mathcal{G} =< \mathcal{V}, \mathcal{E}, \Omega, \pi >$ where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of edges, Ω is a set of labels, and $\pi : \mathcal{V} \cup \mathcal{E} \rightarrow \Omega$ is a function that assigns labels to vertices and edges. Within the data graph, the vertices $\mathcal{V}$ represent different objects in a file system such as users, folders, files and data chunks. The edges $\mathcal{E}$ indicate a variety of relationships among different objects which can be distinguished by a set of labels Ω. The labels also facilitate the process of path-oriented search, e.g., to find all data chunks of one file, or to find all the files of one user. Furthermore, we define $\mathcal{N} \subseteq \mathcal{V}$ as the set of data nodes which store the raw data in $\mathcal{G}$. We aim to distribute data nodes $\mathcal{N}$ to different CSPs in terms of storage protocols.

4.3.3 CSP Model

The cloud storage provider (CSP) model in this chapter includes both user and system specific weights. User-specific weight to each cloud can be assigned either by StoreSim (the default) or by users in terms of their preferences, e.g., the fraction of data they want to store on a particular cloud, the trust that the user has in a CSP or the general reputation of the provider. Meanwhile, the system specific weight can be assigned by StoreSim to evaluate different CSPs in terms of cost, quota, network performance, etc. In this chapter, we model user-specific weight as the *storage load*, i.e., the ratio of the total size of data stored on a cloud to the size of entire data of the user, while the system-specific weight is modeled as *prior knowledge* of a CSP, i.e., the set of data nodes which have been stored on it. Thus, the amount of prior knowledge of a CSP increases with the number of data nodes stored on it. We assume that the knowledge is *unforgettable*, i.e., the knowledge of a data node will not be removed even when the data node is removed from the cloud[2]. To sum up, a CSP $s \in \mathcal{S}$ in StoreSim is parameterized by two factors

[2]This is a necessary assumption since providers such as Dropbox also do not actually delete user's data immediately.

$< u, v >$ where u is a storage load factor while v indicates the prior knowledge of the CSP.

4.3.4 Storage Protocol

In essence, the storage protocol is a set of constraints or cost functions to reduce the information leakage on data distribution across multiple clouds. Motivated by our example in Section 4.1.1, the protocol in StoreSim is to store similar chunks on the same cloud, thereby reducing information leakage to each individual CSP. In the following, we firstly define information leakage for a pair of data nodes.

Definition 1. *(Pairwise Information Leakage).* Given a set of data nodes $\mathcal{N}$ in data graph $\mathcal{G}$, we define $\mathcal{L}_p : \mathcal{N} \times \mathcal{N} \rightarrow \mathcal{R}$ as the pairwise information leakage function. For any pair of data nodes $n_i, n_j \in \mathcal{N}, \mathcal{L}_p(n_i, n_j)$ computes the pairwise information leakage of two nodes.

$\mathcal{L}_p$ can measure the information leakage in terms of either syntactic or semantic way. For example, the information leakage can be measured as the dissimilar information (i.e., new information) based on similarity measures or as the information gain based on entropy measures. In this chapter, we only model information leakage based on syntactic similarity. Specifically, we use delta Jaccard similarity of two sets, as our information leakage function [3].

$$\mathcal{L}_p(n_i, n_j) = J_\Delta(\sigma(n_i), \sigma(n_j)) = 1 - |\frac{\sigma(n_i) \cap \sigma(n_j)}{\sigma(n_i) \cup \sigma(n_j)}| \tag{4.1}$$

where the function $\sigma(\cdot)$ will convert a data node into a set (i.e., representing a data node as a set of words). It follows immediately that if a CSP gets a new data node that is a duplicate of an existing data node on that cloud (i.e., Jaccard similarity between them is 1.0), there will be no leakage due to lack of new information. In the opposite case, the first data node stored in a cloud is a totally new node, which we define the information leakage of the first data node as a constant $\mathbf{1}$. It can be interpreted that all the information in the first data node at a CSP is leaked.

In addition, we also define a storage plan as a mapping from data nodes to different CSPs, which is defined as:

Definition 2. *(Storage Plan).* Given a set of data nodes $\mathcal{N}$ in the data graph $\mathcal{G}$ and a set of CSPs $\mathcal{S}$, a storage plan $\mathcal{M} : \mathcal{N} \rightarrow \mathcal{S}$ is a mapping of each data node $n \in \mathcal{N}$ to a CSP $s \in \mathcal{S}$.

The storage plan can be generated in terms of users' preference and QoS factors. For example, the storage plan based on round robin makes a good balance of the storage load among different CSPs. In this chapter, we will evaluate the goodness of storage plan with respect to the information leakage. The goodness of storage plan is defined as:

Definition 3. *(Goodness of Storage Plan).* Given a set of data nodes $\mathcal{N}$ in the data graph, a pairwise information leakage function $\mathcal{L}_p$ and a storage plan $\mathcal{M}$, we define $G_\mathcal{M}(\mathcal{N}, \mathcal{M}, \mathcal{L}_p) \in \mathcal{R}$ as the goodness function of storage plan $\mathcal{M}$.

From the Definition 3, we can see that the goodness of storage plan depends on the pairwise information leakage function $\mathcal{L}_p$ and the storage plan $\mathcal{M}$. Thus, an interesting question is whether there exists an optimal storage plan with respect to a given information leakage measure. We can formulate this information leakage optimization problem as:

[3] Other appropriate similarity functions can also be employed depending on the data structure and application requirements

Definition 4. *(Information Leakage Optimization Problem).* Given a set of data nodes $\mathcal{N}$ in the data graph, a pairwise information leakage function $\mathcal{L}_p$ and a storage plan $\mathcal{M}$, the information leakage optimization problem is to find the optimal storage plan with minimal information leakage $\mathcal{M}^* = \text{argmin}_{\mathcal{M}}\, G_{\mathcal{M}}(\mathcal{N}, \mathcal{M}, \mathcal{L}_p)$

In this chapter, we provide an approximate algorithm for addressing this problem. We first discuss how to efficiently measure pairwise information leakage in Section III and then in Section IV we propose a storage plan that places similar chunks together in a multicloud environment.

4.4 Efficient Measurement of Pairwise Information Leakage

We define the pairwise information leakage as delta Jaccard similarity, as is shown in Equation 4.1. For each pair of data nodes (chunks), we convert the data nodes as sets of words and compute the Jaccard similarity. However, the *set* operations for measuring pairwise similarity can be quite expensive [103], even assuming small-sized chunks, given that the number of pairs increases quadratically as the number of chunks increases. Thus, we need an efficient algorithm to compute the Jaccard similarity with less computation and storage overhead. In the following, we first introduce the background of MinHash algorithm [103, 104, 105], which provides a fast way to compute Jaccard similarity, and explain why we cannot apply the existing approaches directly. Next we present BFSMinHash, a Bloom filter sketch for MinHash in order to reduce storage overhead.

4.4.1 MinHash Background

MinHash uses hashing to quickly estimate the Jaccard similarity of two sets, $J(S_1, S_2) = |\frac{S_1 \cap S_2}{S_1 \cup S_2}|$. It can be also interpreted as "the probability that a random element from the union of two sets is also in their intersection":

$$Prob[min(h(S_1)) = min(h(S_2))] = |\frac{S_1 \cap S_2}{S_1 \cup S_2}| = J(S_1, S_2)$$

where h is the independent hash function and $min(h(S_1))$ gives the minimum value of $h(x), x \in S_1$. Therefore, we can choose a sequence of hash functions $h_1, h_2, \cdots, h_k$ and compute the minimum values of each hash function as MinHash signatures.

$$Sig_1 = \{min(h_i(S_1))|i = 1, \cdots, k\}$$

$$Sig_2 = \{min(h_i(S_2))|i = 1, \cdots, k\}$$

It follows that Jaccard similarity of two sets is approximated as $|Sig_1 \cap Sig_2|/k$. However, MinHash with many hash functions needs to compute the results of multiple hash functions for every member of every set, which is computationally expensive. In this work, we adopt a variant of Minhash which avoids the heavy computation by using only a single hash function. Instead of selecting only a single minimum value per hash function, the signature of MinHash with single hash function h will select the k smallest values from the set $h(S)$, which is denoted as $min_k(h(S))$. In this way, we have

$$Sig_1 = \{min_k(h(S_1))\}$$

$$Sig_2 = \{min_k(h(S_2))\}$$

Algorithm 4.1: Bloom-filter sketch for MinHash

```
Require: byte[] chunk: byte stream of a data chunk
Ensure: byte[] signature
1: List<byte[]> shingles = ByteSegment(chunk,size);
2: maxHeap ← store k smallest values in a max heap
3: for each shingle : shingles do
4:     fingerPrint = hashFunction(shingle);
5:     maxHeap ← fingerPrint
6: end for
7: BloomFilter bf; //implement with a single hash function
8: for each fingerPrint : maxHeap do
9:     bf.add(fingerPrint);
10: end for
11: byte[] signature = bf.toByteArray();
12: return signature
```

Thus, a random sample of $S_1 \cup S_2$ can be represented as:

$$X = \{min_k(h(S_1 \cup S_2))\} = min_k(Sig_1 \cup Sig_2)$$

The Jaccard similarity is estimated as $|X \cap Sig_1 \cap Sig_2|/k$. For Minhash algorithm, to compute the similarity for a pair of data nodes, we only need to store an array of MinHash signatures rather than storing the whole data. Although it reduces the storage cost greatly, it can still be heavy given the huge number of data nodes. Suppose that each hash function generates a signature of 64 bits and k is 64, the storage cost of each data node is about 512 bytes. If we have about two million chunks, the overhead of storing the signatures is 1 Gigabyte. Thus, we need a compact representation of these MinHash signatures to reduce the storage overhead. Previous work [106] proposed *b-bit* MinHash which only stores b lowest bits of each signature computed by different hash functions to reduce the storage space. However, this approach does not work for the MinHash with a single hash function since all the signatures are computed by the same hash function. Instead, we design BFSMinHash, a Bloom-filter sketching scheme for Minhash, which uses a single hash function. BFSMinHash exploits the space efficient feature of Bloom filter, thus reducing the storage overhead.

4.4.2 Bloom-filter Sketch for MinHash

Similar to the fingerprints in data deduplication, we expect an algorithm to generate the signature with a relatively small and fixed size for each data node. Our proposed BFSMinHash algorithm employs a Bloom-filter with a single hash function to sketch MinHash signatures. Algorithm 4.1 shows three steps in BFSMinHash: *shingling* (line 1), *fingerprinting* (line 2-6) and *sketching* (line 7-11). The input is a byte stream of a data chunk and the output is a fix-sized similarity-preserving signature of this chunk.

Firstly, we convert each data chunk to a set of shingles which are contiguous subsequences of tokens. The process of shingling is to tokenize the byte stream into a set of shingles. For example, if the input is "abcde" and the size of a shingle is 2, the set of shingles is {ab, bc, cd, de}. From this perspective, we only consider the similarity in a syntactic way [107] rather than in a semantic way. In other words, we do not consider the difference between the fruit apple and the company Apple. Then, for each shingle,

we will compute its fingerprints by MinHash. We use a maximum heap with the fixed-size of k to save k smallest MinHash fingerprints for each data node. It only takes $O(1)$ to get the maximum value of all k values in a maximum heap. Only when a new fingerprint is less than the maximum value stored in the heap, it will be added to the heap and the current maximum in the heap will be removed. From the shingling and fingerprinting steps, we can see that the time complexity of our algorithm is linear in the total length of data chunks. Finally, sketching based on Bloom-filter will convert the MinHash fingerprints into a fixed size signature. The Bloom filter is a space efficient data structure which can be used to test whether an element is in a set. However, when we adopt Bloom filter, we have to tolerate its effect of false positives. The rate of false positives is computed as $(1 - e^{-nk/s})^n$, where s is the size of Bloom filter, k is expected number of elements that will be added in Bloom filter and n is the number of hash functions [108]. For example, if we implement a Bloom filter with size of 512 bits and k is 64, the optimal number of hash functions is 1 with a false positive rate of 11.7%. In our case, we aim to keep the size of Bloom filter as small as possible and therefore the Bloom filter in our BFSMinHash algorithm always employs a single hash function. The final output of Algorithm 4.1 is a signature with the same size as the Bloom filter. In this way, computing similarity of two data nodes is converted to compute the similarity of two bloom filters. Given two signatures x, y, the Jaccard similarity is

$$J(x, y) = \frac{\sum_i (x_i \wedge y_i)}{\sum_i (x_i \vee y_i)} \tag{4.2}$$

where x_i, y_i is the ith bit of x, y, and $\wedge$, $\vee$ are bitwise *and, or* operators respectively. Later we will evaluate approximate errors of BFSMinHash, which are caused by both MinHash and Bloom filter, in Section 4.6.

4.5 Generating Multicloud Storage Plan

Based on the pairwise information leakage measured by BFSMinhash algorithm, the next step is to generate the storage plan, as shown in Definition 2, with respect to the information leakage. Before we present our storage plan generation algorithm, we need to introduce a goodness function to quantify the quality of a storage plan.

4.5.1 Goodness of Storage Plan

The goodness function of storage plan is evaluated based on the pairwise information leakage, as it is defined in Definition 4. Recall from Equation 4.1, the pairwise information leakage measures how much new information will be leaked when a pair of data nodes are stored in the same cloud. Thus, it is essential to find the pairs of data nodes with minimal information leakage. In order to measure the goodness of a storage plan, we introduce a metric called *relative information leakage* (RIL), which is defined as the average of minimal pairwise information leakage among all the data nodes in a storage plan. For example, in our motivating example in Section 4.1.1, cloud S_2 stores three data nodes for a total of $\binom{3}{2}$ pairs, $\{(C2, C5), (C5, C8), (C2, C8)\}$. Suppose $\{\mathcal{L}_p(C2, C5) = 0.25, \mathcal{L}_p(C5, C8) = 0.15, \mathcal{L}_p(C2, C8) = 0.1\}$, we have the information leakage of first data node $C2$ as constant **1** while the minimal pairwise information leakage for $C5, C8$ is 0.15 and 0.1, respectively. Thus, the RIL of data nodes stored in S_2 is the average minimal pairwise information leakage $(1 + 0.15 + 0.1)/3 = 0.416$.

Formally, given an individual CSP $s_i = (u_i, v_i) \in S$ in a storage plan $\mathcal{M}$, the RIL of all data nodes stored in s_i is formulated as:

$$RIL_i = \frac{1}{|v_i|}(\mathbf{1} + \sum_{l=2}^{|v_i|} \mathcal{L}_{min}(n_l, n_k)), \tag{4.3}$$

$$s.t. \quad v_i = \{n \in \mathcal{N} | \mathcal{M}(n) = s_i\}, \tag{4.4}$$

$$l \neq k, n_l, n_k \in v_i \tag{4.5}$$

where $\mathbf{1}$ is the information leakage for the first data node and $\mathcal{L}_{min}(n_l, n_k)$ returns the minimal pairwise information leakage, $\mathcal{L}_p(n_l, n_k)$, for n_l by searching the node $n_k, l \neq k$, which is most similar to it. v_i in Equation 4.4 represents prior knowledge and is modeled as the set of data nodes stored in s_i. Since we have $\mathcal{L}_p \in [0, 1]$, it follows that $RIL_i \in [\frac{1}{|v_i|}, 1]$. In the extreme case where all the data nodes stored in a CSP are the same, the RIL is $\frac{1}{|v_i|}$, which means the actual information it has obtained equals to the information of one node. In other words, a good storage plan, which can effectively detect the similar chunks and distribute them to the same cloud, has a low RIL value. Based on this, we can compute the RIL for a storage plan $\mathcal{M}$ as the weighted average of the RILs of all CSPs:

$$RIL_{\mathcal{M}} = \sum_{i=1}^{|\mathcal{S}|} u_i * RIL_i \tag{4.6}$$

where u_i is the normalized user-specific weights of CSPs such that $\sum_{i=1}^{|\mathcal{S}|} u_i = 1$. In this way, the information leakage optimization problem with respect to RIL is to find an optimal storage plan with minimal relative information leakage to each CSP.

4.5.2 Clustering for Storage Plan Generation

In Equation 4.3, $\mathcal{L}_{min}$ needs to find the pairs with the minimal information leakage. This search problem is challenging when the number of pairs increases quadratically. Suppose we have 100,000 data nodes, the number of pairs will be as high as 5 billion $\binom{100,000}{2}$. Thus, we need to design an efficient search algorithm to find data pairs with minimal information leakage.

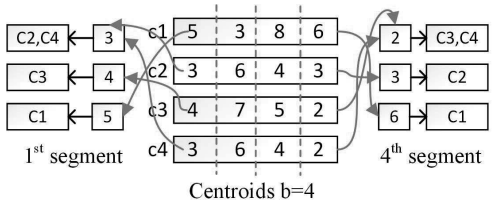

Figure 4.3: ClusterIndex for Centroids with b=4 Segments

Inspired by clustering problems [109], we propose a storage plan generation algorithm, *SPClustering*, to group similar data nodes. We define a data node as the *centroid* when no existing data node has low pairwise information leakage with it. In practice, we define a leakage threshold, according to which a data node becomes a centroid if all its pairwise information leakage with other nodes are greater than this threshold. In other words, a centroid represents all data nodes which are similar to it. Given any

new data node, we only compute its pairwise similarities with a set of centroids, which largely reduces the number of pairs. Moreover, we build the *ClusterIndex* among the centroids to further prune the search space. A single index entry in ClusterIndex points to a set of similar centroids, which is similar to the Bitmap index in traditional databases [110]. Specifically, suppose the size of signature generated by BFSMinHash algorithm is s bits, we divide the signature into b segments with the length of each segment as s/b. We will use each segment as the key in hash function and therefore, all the signatures with the same key will be hashed together. For example, as is shown in Figure 4.3, when the key is the value of first segment, $c2$ and $c4$ are hashed to the same index entry for they share the same value of first segment. Those signatures are more likely to be similar to each other since they already share one same segment. Recall from Section 4.2, the number of elements sampled by BFSMinHash is k, which means its signature based on Bloom filter is at most with k bits set to one. If we cannot search any similar node from the ClusterIndex with b segments for a given node, that means there are at least b bits different from the given node with all the centroids. Based on Equation 4.2, it implies that there is no centroid that has Jaccard similarity with the given node larger than $(k-b)/(k+b)$. For example, if k is 64 and we divide the signature into 8 segments, the ClusterIndex can efficiently search all the similar centroids with similarity higher than 77.8%. Thus, in order to find centroids with less or more similarity, we need to respectively increase and decrease the value of b (the number of segments).

Algorithm 4.2: Generating storage plan based on clustering

Require: $\mathcal{N}$: a set of data nodes, $\mathcal{S}$: a set of CSPs
Ensure: $map \in \mathcal{M}$ storage plan
1: Build ClusterIndex for all centroids
2: **for** each $x : \mathcal{N}$ **do**
3: **for** each $s : \mathcal{S}$ **do**
4: c = **getCandidateSet**(x, s) //pruning
5: $loss \leftarrow \frac{1}{|c|} \sum_{y \in c} \mathcal{L}_p(x, y)$
6: **end for**
7: $min\_loss \leftarrow$ find s with minimal loss
8: **if** $min\_loss > threshold$ **then**
9: assign x based on weights of CSPs
10: add x as a centroid and build ClusterIndex for x
11: **end if**
12: map.put(x,s)
13: **end for**
14: **return** map

Algorithm 4.2 shows three main steps of how to generate a storage plan. Firstly, in the initialization, the algorithm builds the ClusterIndex for a set of centroids online. We do not persist the ClusterIndex to reduce the storage overhead. The cost of building ClusterIndex is acceptable, which takes about 400 milliseconds for 100 thousand centroids. Then, we will find the cloud with the minimal information leakage based on candidate set for each new data node. The candidate set is queried based on ClusterIndex. Finally, if the minimal information leakage is still larger than the threshold, we will assign this node only based on the weights of CSPs. Also, the node will be labeled as the centroid and be indexed on the fly.

4.6 Experimental Evaluation

In this section, we first introduce the implementation of StoreSim and the two datasets used for evaluation. Then we evaluate the performance of two algorithms, BFSMinHash and SPClustering. Finally, we analyze the time cost introduced by the leakage measure layer in StoreSim.

4.6.1 Implementation

We have implemented the StoreSim prototype using Java, and it includes both basic components (such as chunking, data deduplication, bundling and encryption/decryption), and featured components including LMLayer and CMLayer. In the LMLayer, we implement the algorithms described in the previous sections, while the CMLayer enables StoreSim to communicate with multiple CSPs. All those components are optional, i.e., users can opt to use the LMLayer and encryption at the same time or they can disable any of them. StoreSim is also pluggable, i.e., StoreSim provides APIs for developers to add their modules for different encryption or information leakage measures. For example, StoreSim employs the BFSMinHash algorithm but developers can replace it with the SimHash algorithm based on Hamming distance [111]. StoreSim employs the common fixed-size chunking with a maximum chunk size of 512 KB. The chunk is identified by *SHA-1* signature, which is also used for data deduplication. The small chunks can be bundled as a ZIP file to minimize the network transmission overhead. Succinctly, before the chunk is synchronized, it can be measured for leakage optimization, encrypted, and bundled for better network transmissions.

The synchronization of StoreSim is based on *rsync*-like protocols [97], which only synchronizes the new chunks (identified by SHA-1 signatures) between two copies. All the metadata, which is organized as data graph, are stored in a MySQL database. To deal with the heterogeneity in the different data models of different clouds, StoreSim employs CMLayer to enable cloud interoperability in a syntactic way. CMLayer provides the uniform APIs such as initialize, connect, upload, download and delete, for different CSPs. With this abstraction, the user can move their data across different CSPs in a transparent way, thereby alleviating vendor lock-in problems. We have implemented for three public storage clouds: Dropbox, Google Drive, and Amazon S3. All the communications between StoreSim and public CSPs are using APIs supplied by those CSPs. We also support the synchronization of files to the local FTP servers. The metadata server is deployed on our local server machine and the evaluation is conducted on a personal client machine with Intel i7-2640M CPU and 4GB memory.

4.6.2 Dataset

For the evaluation, we aim to find such data which has undergone several modifications, and thus results in many similar chunks. This can serve as a model for the modifications that users make in the cloud storage services. Wikipedia and Github are two such data sources that contain web pages and files which are reviewed and modified multiple times. Thus, we crawled two datasets from Wikipedia and Github, respectively. The Wikipedia dataset contains a total of 2197 web pages and each web page has a maximum 49 revisions. For each web page, the crawler only stores the text that is extracted from HTML files. The total size of the dataset is 1.2 GB. The size of each webpage is relatively small, which ranges from 29 Bytes to 118 KB with an average size of 11KB. The Github dataset contains the United States

code[4] spanning 56 files. The files in this dataset are much larger than those in the Wikipedia dataset, in the range of 47.7KB to 50MB with an average size of 5.3 MB. The files in this dataset have a maximum of 8 modifications and the total dataset size is 2.1 GB. Thus, we observe that the data chunks generated by Wikipedia dataset are small in size with maximum chunk size of 118 KB, but great in number (91,929) while those generated by Github dataset are bigger in size with maximum size of 512KB but are less in number (4,274).

4.6.3 BFSMinHash

In this part, we will evaluate the performance of our BFSMinHash algorithm and answer two questions:

- what's the approximation errors and effectiveness of our proposed BFSMinHash?
- Is the information leakage measured by BFSMinHash trustable?

Approximation Errors. We implement BFSMinHash based on 64 bits *Murmur* hash function [112] and thus each MinHash signature is 64 bits. In BFSMinHash, we have to further fix three parameters: the shingle size l, the sampling size of MinHash k and the Bloom filter size s. The approximation error of MinHash algorithm is caused only by sampling and shingle size while that of our BFSMinHash is due to all the parameters. The larger the size of sampling and Bloom filter, the closer that our algorithm approximates to the actual Jaccard similarity. However, we have to balance the tradeoff between approximation errors and storage cost. We seek to determine the suitable parameters for BFSMinHash for our subsequent experiments, by first comparing the performance among five settings of our algorithm: 1) MinHash-10-128; 2) MinHash-10-64; 3) BFSMinHash-10-64-512; 4) BFSMinHash-10-64-256; 5) BFSMinHash-8-64-256, where the numbers in the name correspond to the value of l, k and s (only for BFSMinHash). Among different settings, MinHash-10-128 theoretically has the best performance since it has the largest sampling and shingle size and no sketch. Given our goal is to evaluate the approximation error between our BFSMinHash and MinHash algorithm, we select MinHash-10-128 as the baseline and compare its performance with that of the other four algorithms. In addition, to evaluate the performance of different algorithms, we define approximate error as the root mean square error (RMSE) between the result of MinHash-10-128 and that of the algorithm under comparison, i.e., $RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}}$.

In this part, we randomly select two small sample datasets from Wikipedia and Github datasets due to huge number of pairs present in the original datasets. We select 400 pages from Wikipedia and 30 files from Github in which each page or file in these samples has 4 revisions. After chunking by StoreSim, the number of data chunks for both samples is around 1600. Thus, the total number of pairs is around 1,279,200 ($\binom{1600}{2}$). From Figure 4.4, we can see that MinHash-10-64 without sketching outperforms the other three for both datasets. The sampling size of MinHash-10-64 is 64, i.e., it will select 64 smallest MinHash signatures. The storage cost for each chunk is 64*64 bits = 512 Bytes. We can observe that BFSMinhash algorithms with the shingle size of 10 are better than that with the shingle size of 8. This is because a longer shingle size decreases the probability of a given shingle appearing in any document. In addition, we also observe that approximate error is influenced by the ratio of the sampling size to the Bloom filter size. As is shown in Figure 4.4, the performance of BFSMinhash-10-64-512 is better than BFSMinHash-10-64-256 by about 6%. However, the storage cost of BFSMinHash-10-64-256 (32 Bytes per chunk) is only half of BFSMinHash-10-64-512 (64 Bytes per chunk).

[4] https://github.com/divegeek/uscode

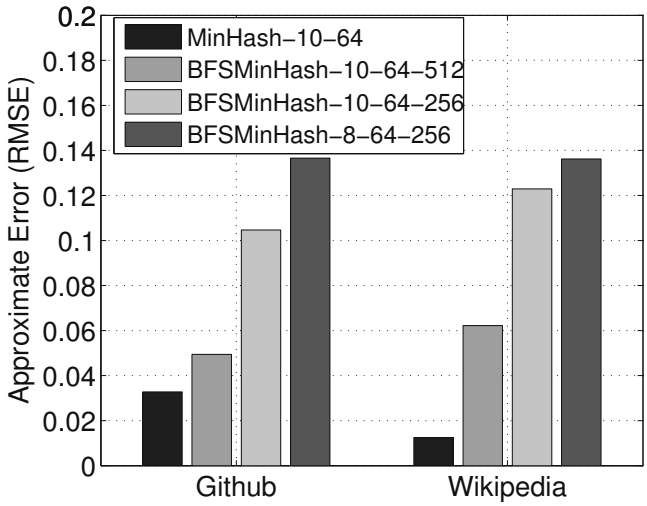

Figure 4.4: Effect of parameters with MinHash-10-128 as baseline

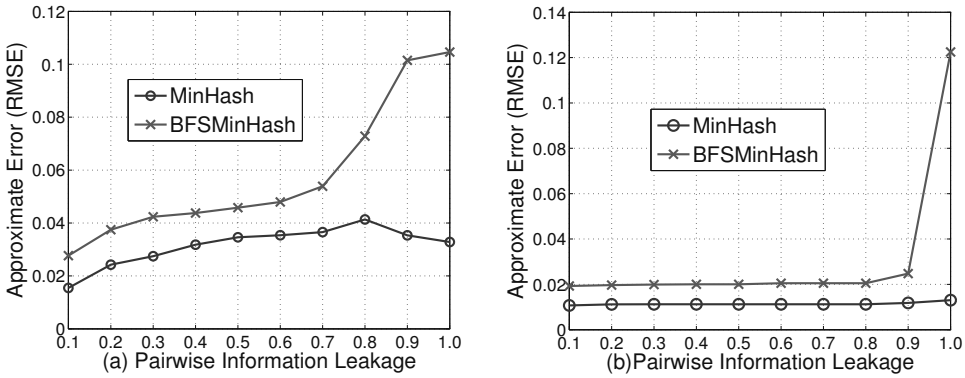

Figure 4.5: Approximate Errors by groups for (a) Github and (b)Wikipedia

Considering the tradeoff between storage cost and approximate errors, in StoreSim we adopt the setting of BFSMinHash-10-64-256. We observe that overall approximate errors of BFSMinHash-10-64-256 are about 10.4% for Wikipedia and 12.2% for Github. Thus, an immediate question is that are these approximation errors of BFSMinHash-10-64-256 tolerable to find the pairs with the minimal information leakage? We will answer this question in the next group of experiments. In the following, we use BFSMinHash to refer to BFSMinHash-10-64-256 while MinHash refers to MinHash-10-64.

Effectiveness of BFSMinhash. In the last experiment, we evaluated approximate errors based on all the pairs. In fact, the primary goal of our algorithm is to identify those pairs which have minimal information leakage and put them in the same cloud. Thus, we are more interested in approximate errors of the pairs with the minimal information leakage. Put bluntly, we are interested in those pairs of nodes whose information leakage is low, say 0.3, rather than those whose information leakage is very high, say 0.8, since these do not serve our needs of placing similar nodes on the same CSPs.

Therefore, in this set of experiments, we divide pairs into ten groups in terms of their pairwise

information leakage, where the first group is all the pairs with information leakage less than 0.1 while the second group is set of pairs with information leakage less than 0.2, and so on. The approximate errors of different groups are shown in Figure 4.5. It is interesting to discover that the performance of BFSMinhash is highly close to the MinHash algorithm for groups 1-7 of Github dataset and groups 1-9 of Wikipedia dataset. For the Github dataset, the performance of BFSMinhash degraded dramatically after the information leakage is larger than 0.7 while for the Wikipedia dataset, the performance of BFSMinhash remains stable till the information leakage is 0.9. The dramatic increase in approximation errors of the pairs with large information leakage means that our algorithm is not very accurate for the pairs with low similarities. However, as stated earlier, in practice, we are targeted to identify the pairs with low information leakage (or high similarity). Therefore, we can safely state that our BFSMinhash algorithm is effective enough to meet our demands since it identifies pairs with information leakage as high as 0.7 with low error. To conclude and to answer the question raised by the last set of experiments, the results clearly show that our BFSMinHash is almost as effective as MinHash in identifying the pairs with the minimal information leakage while it can reduce the storage cost to *1/16* of MinHash.

Is the measured information leakage trustable? Given the pairwise information leakage computed based on our BFSMinhash algorithm, another question is that to what extent we can trust those values. To answer this question, we have to obtain the ground truth of pairwise information leakage. Instead of using Human evaluation [111], we compute pairwise similarities among all the data chunks using the Gensim library[5] for calculating cosine similarity between documents based on TF-IDF weights. We assume that the ground truth of pairwise information leakage is highly close to the result computed by Gensim algorithm[6]. Based on the Gensim result, we can generate the *relevant set*, which is denoted as R, as the set of pairs whose information leakage is less than 0.2. In the next step, we will query a set of pairs from the result generated by our BFSMinHash algorithm, denoted as *search set* S, with information leakage threshold ranging from 0.1 to 0.7. i.e., we query all the pairs with information leakage less than 0.1, all the pairs with information leakage less than 0.2, and so on. Therefore, we can calculate the precision, i.e., the fraction of searched pairs that are relevant to all the searched pairs, and recall, i.e., the fraction of the searched pairs that are relevant to all the relevant pairs, based on the search set S and relevant set R : $Prescion = \frac{|S \cap R|}{|S|}$, and $Recall = \frac{|S \cap R|}{|R|}$.

Figure 4.6 shows the results of both datasets. It clearly shows the tradeoff of precision and recall under different information leakage thresholds. For the Github dataset, choosing threshold as 0.4 achieves both good precision(0.85) and recall(0.75). To our surprise, for the Wikipedia dataset, both the precision and recall can be as high as 99.5% when we set the threshold to 0.3. That means the search set generated by our BFSMinhash algorithm is almost the same as the relevant set which is generated by Gensim algorithm. Therefore, we can safely reach the conclusion that the information leakage computed by BFSMinHash algorithm can be almost as good as that of GenSim by choosing an appropriate threshold. In this case, we can set the threshold as 0.4 for both datasets, which is the point in both Figure 4.6(a) and (b) where a good balance between precision and recall is achieved. Furthermore, we can also observe from Figure 4.6, that the performance of BFSMinhash and Minhash are similar for low thresholds. This further confirms the effectiveness of our BFSMinHash algorithm in identifying the data pairs with low information leakage.

[5]http://radimrehurek.com/gensim/

[6]We note that we cannot apply Gensim to StoreSim since it is too time expensive. For the given data, it took us more than 24 hours to compute the information leakage for all pairs on a server machine.

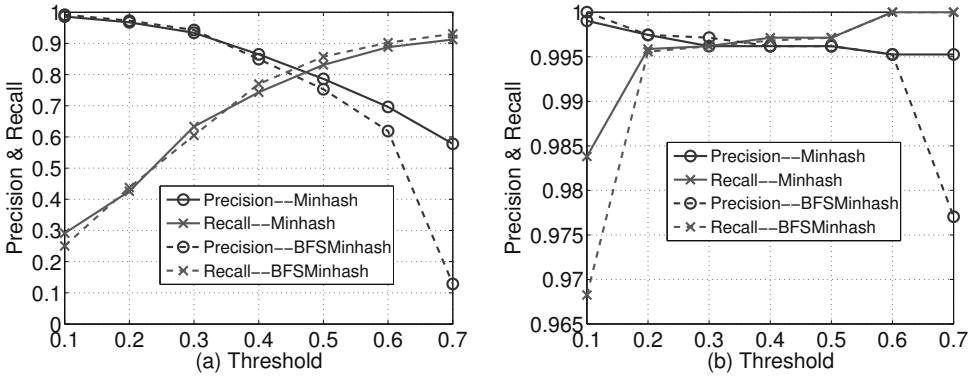

Figure 4.6: Precision and recall by varying threshold for (a) Github and (b) Wikipedia

4.6.4 SPClustering

In this part, we will evaluate the performance of our storage plan generation algorithm SPClustering. Besides the metric of RIL as defined in Section 5.1, we further define a new metric, *information density* (InfoD) from the perspective of entire dataset. The InfoD of a CSP is defined as the ratio of the information it has stored to the entire information in the whole dataset. Given a CSP $s_i = (u_i, v_i) \in S$, we further denote the set of data nodes which are also centroids stored in s_i as s_i^c and the InfoD of s_i is computed as:

$$InfoD_i = \frac{|v_i^c|}{\sum_{j=1}^{|S|} |v_j^c|} \tag{4.7}$$

where $\sum_{j=1}^{|S|} |v_j^c|$ denotes the total number of centroids in a dataset. From Equation 4.7, we approximate the total information in a dataset to that information in its centroids since the centroid represents all data nodes which are similar to it. Base on this, the InfoD of a storage plan $\mathcal{M}$ for a dataset is computed as the weighted average InfoD of each CSP:

$$InfoD_{\mathcal{M}} = \sum_{i=1}^{|\mathcal{S}|} u_i * InfoD_i \tag{4.8}$$

For example, consider all CSPs with equal normalized weights of $\frac{1}{|\mathcal{S}|}$. Here the optimal case of storage plan ensures that $InfoD = \frac{1}{|\mathcal{S}|}$, with every cloud obtaining $\frac{1}{|\mathcal{S}|}$ of total information, (i.e., $InfoD_{\mathcal{M}} = \frac{1}{|\mathcal{S}|}$); while the worst case is $InfoD = 1$ with every cloud obtaining all the information in the dataset, (i.e., $InfoD_{\mathcal{M}} = 1$). Thus, we can see that the higher the InfoD is, the more information are leaked to each SDC.

In the following, we will evaluate the goodness of storage plan generated by our SPClustering in terms of both RIL and InfoD to answer three questions:

- What's the impact of user's modifications of data on the information leakage?
- Is there any effect on the number of CSPs on which the users distribute their data?
- How does the weight specified to different CSPs by users influence the information loss?

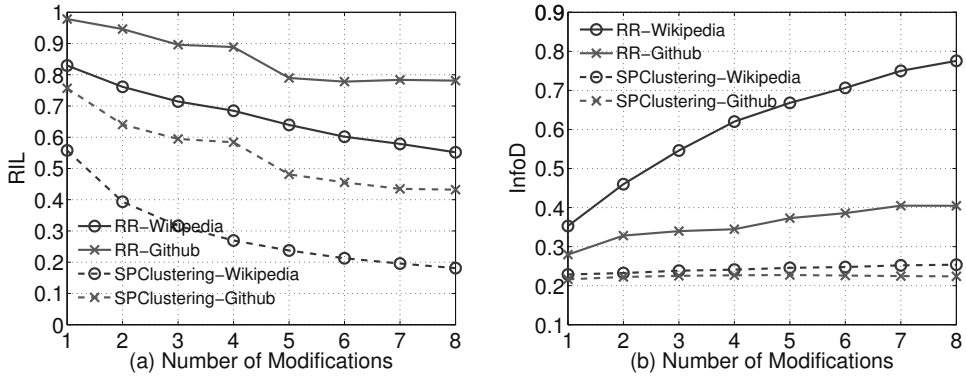

Figure 4.7: Effect of modification numbers on (a) RIL and (b) InfoD

Number of modifications. In this set of experiments, we have five CSPs with equal weights. After each modification, the dataset will be synchronized to clouds using delta encoding. The more modifications a dataset has undergone, the more resultant similar chunks. Figure 4.7 shows the influence of number of modifications on information leakage of storage plans generated by both SPClustering and Round Robin (RR) algorithms. It clearly shows that SPClustering outperforms RR greatly for both the Wikipedia and Github datasets. From Figure 4.7 (a), we can observe that with the increase in number of modifications, the RIL of SPClustering decreases, by about 30% (from the first modification to the last), much more quickly than RR, which decreases by only about 16%. The decrease of RILs implies that modifications on a dataset brings about more similar chunks. Our SPClustering algorithm is much more effective than RR to place those similar data chunks with the minimal information leakage in the same cloud. In Figure 4.7 (b), we can observe that RR without clustering the similar data nodes leaks the information in the dataset quickly, for the infoDs of RR increase to about 80% and 40% for Wikipedia and Github, respectively. Recall that the number of data chunks in Wikipedia dataset is much larger than that in Github, which also brings about much more similar data chunks. Thus, under RR without optimization on data chunks distribution, we can observe that Wikipedia leaks information much quicker than Github. On the other hand, InfoDs of our approach almost remains stable (around 22%, 25% for Github and Wikipedia, respectively), which indicates that our approach prevents the information leakage effectively. The reader may recall that the files in Wikipedia dataset have undergone a maximum of 49 modifications while that of Github a maximum of 8 modifications. Thus, we only compare the first 9 versions of Wikipedia dataset to that of whole Github dataset in Figure 4.7. If we evaluate the whole Wikipedia dataset with 49 modifications, the final RILs of Wikipedia decreases to 9.7% while InfoDs of Wikipedia increase to 31%. Thus, we can conclude that our approach greatly prevents information leaked in the process of data synchronization.

Number of CSPs. In this set of experiments, we fix the number of modifications to 8 and vary the number of CSPs from 2 to 5. All CSPs in the experiments have the same weight. In Figure 4.8(a), we can see that the RILs of SPClustering are almost stable for both datasets while those of RR increase steadily. The stable RIL implies that SPClustering algorithm is effective to prevent information leakage by putting the pairs with the minimal information leakage together regardless of the number of CSPs. As for RR, with the increase in number of CSPs, the probability of putting data pairs with minimal information leakage in the same cloud decreases, thereby leading to increased RIL. In Figure 4.8(b), we

can observe that InfoDs of our approach decreases as the number of CSPs increases. This is the benefit of multicloud environment and the more CSPs there are, the less the data obtained by each cloud. However, we can observe from Figure 4.8(b) that a CSP under RR without considering information leakage can obtain more than 70% of entire Wikipedia dataset and 40% of Github dataset even with a total of 5 CSPs while SPClustering can achieve near-optimal value with respect to InfoD. For all cases, we observe that SPClustering improves the InfoD by about 60% and 45% for Wikipedia and Github respectively, compared to RR, which means SPClustering prevents about 60% of total information in Wikipedia from being leaked.

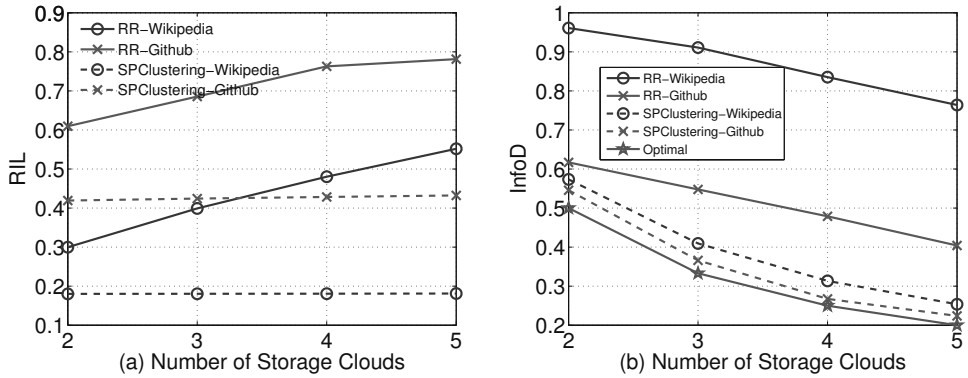

Figure 4.8: Effect of CSP numbers on (a) RIL and (b) InfoD

User-specific weight. In practice, users may have different weights for different CSPs. In StoreSim, the weight of each CSP not only coordinates the storage load but also the InfoD for preventing the information leaking. In this part, we do the experiments with 5 CSPs with normalized weights. We increase the weight of one CSP from 0.2 to 0.5 while the remaining is evenly distributed to the other four CSPs. Both datasets have 8 modifications and are synchronized to five CSPs for 9 times. Figure 4.9(a) shows the results for the CSP with the varying weight under SPClustering and *weighted round robin* (WRR). The bottom most line represents the expected InfoD in terms of user specified weights. Under SPClustering, it can be clearly observed that users can control the InfoD of a CSP by assigning different weights to it. However, the results also shows that information leakage of WRR for Wikipedia dataset can be as high as nearly 100% when it is assigned to obtain only 50% of the entire data. In other words, under WRR, the CSP ends up obtaining nearly all the information even though the user expected it to obtain only 50%.

ClusterIndex. Finally, we evaluate the pruning efficiency of ClusterIndex employed by SPClustering algorithm. We vary the number of data nodes in ClusterIndex from 2000 to 90,000 in Wikipedia dataset and compare the performance with that of indexing without considering clustering. We only evaluate based on Wikipedia dataset since the number of data chunks in Github is limited. As is shown in Figure 4.9(b), ClusterIndex can reduce the number of searched pairs by 86% without much tradeoff on the precision (about 2.6%, not shown in the figure).

4.6.5 System Attackability Analysis

In this section, we will analyze our StoreSim in terms of system *attackability*, where the attackability of a system is a measure of utility of potential attacks in terms of benefits gained and costs incurred. We

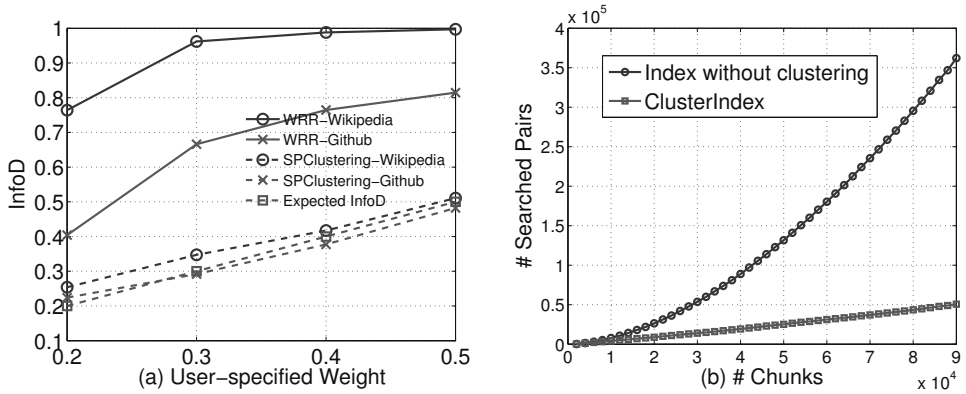

Figure 4.9: (a) Effect of user's weight and (b) Pruning efficiency by ClusterIndex

assume that the attackers maintain a network of resources to establish multiple *attack channels* to launch attacks to the target system. The attack channel can be either the communication links in computer networks (e.g., cyber attack) or the Human connections in social engineering (e.g., insiders attack). We define the average cost of an attack channel as c which can be evaluated in terms of time or money. In the multicloud scenario, instead of a single point of attack to get everything, the attackers have to figure out many things e.g., the number of CSPs that stores the data and the impact on the number of attack channels along with the increase in the number of CSPs, which makes the attack much more complex. To model this complexity, we assume that the attack channels has a polynomial growth rate $\mathcal{O}(n^m)$ with the number of CSPs adopted in a multilcoud system, where n is the number of CSPs and m is denoted as *multicloud effect* factor. Thus, given the average cost per attack channel c, we can define the cost function of an attack as $f(c, n, m) = c * n^m$, which captures that the cost of an attack increases with the number of attack channels. The c is mathematically a scale factor which can incorporate other extra costs due to the diversity factors in different systems. For example, the attacker may take more time and money on attacking an encrypted system.

For different CSPs storing different proportion of information, $w_1, w_2, \ldots$ with probabilities of being compromised in an attack, $p_1, p_2, \ldots$, the expected benefit for attackers is $\sum_{i=1}^{n} p_i * w_i$. Based on this, We define the attackability of a system as a benefit-cost ratio to the attacker.

$$attackability = \frac{\sum_{i=1}^{n} p_i * w_i}{c * n^m}, 0 < p_i, w_i \leq 1 \tag{4.9}$$

We can see that the high attackability of a system means that the attacker can gain high benefits at low costs. The probability p_i can be also interpreted as the risk of information leakage for each SDC. We note that p_i can equal 1 which can happen under the insider attack or coercion from the government. We also note that the attackability of a system is inversely proportional to n^m, which means that the more CSPs a multicloud system has and the higher multicloud effect is, the more arduous a system can be compromised. In the following, we will further discuss four different types of systems: **1) SingleCloud**, a single centralized cloud storage system with $n = 1, w_1 = 1$: in this case, multicloud effector factor m has no impact on the number of attack channels since it is not a multicloud system. The attackability of SingleCloudSys is $\frac{p_1}{c}$, which only depends on the average cost and the probability of CSP being compromised. **2) OptMulCloud**, an optimal multicloud storage system with $n > 1, \sum_{i=1}^{n} w_i = 1$: under this case, we can see that all the information are ideally distributed among the SDCs, corresponding

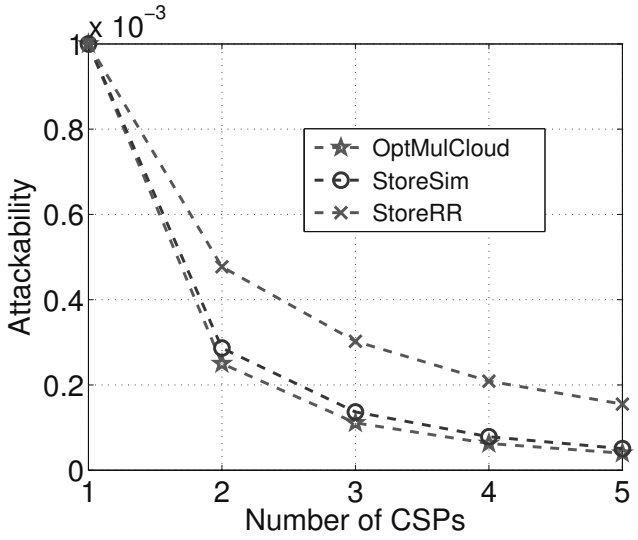

Figure 4.10: Attackability of different systems with c=1, m=2, p=0.001

to the weight of each SDC. **3) StoreSim**, our proposed system approximates to the optimal status by putting all the similar data nodes in the same cloud. **4) StoreRR**, the system with RR data distribution: the system without clustering similar data nodes in the same cloud may leak more information during the distribution. Both StoreSim and StoreRR have the setting that $n > 1, \sum_{i=1}^{n} w_i > 1$.

For the purpose of comparison, we keep three factors c as 1, m as 2 and p as 0.001 unchanged while varying the number of CSPs n from 1 to 5. We assume that all the CSPs are equally reliable with the same probability $p = 0.001$ of being compromised (i.e., the reliability of all the CSPs is 0.999) and all the CSPs are assigned the same weight to get the same amount of information. The values of w_i for StoreSim and StormRR are supplied by values of information density which are derived from our evaluation. Figure 4.10 shows the attackability of four proposed systems. It is clear that our proposed StoreSim, whose performance is near to that of OptMulCloud, is two times more complex than StoreRR to be attacked as the number of CSPs is 5. As for *SingleCloud* system ($n = 1$ in the Figure 4.10), all the information are stored in only one centralized cloud, which leads to a high value of attackability. Furthermore, it is worth mentioning that the attackability of StoreRR with 5 CSPs is higher than that of StoreSim with only 3 CSPs. This is an important result which shows that unplanned data distribution is worse than our scheme even under the presence of comparatively more CSPs.

From our attackability analysis, we can safely conclude that under our proposed StoreSim system, optimized data distribution across the multicloud reduces the risk that *wholesale* information is leaked and makes the attacks on *retail* information much more complex.

4.6.6 Discussion

In this part, we will discuss limitations of our StoreSim from three perspectives.

CPU overhead. It is clear that the client in our system performs more additional work which introduces more computation. To reduce the CPU overhead, we choose MinHash algorithm with a single hash function and propose SPClustering with ClusterIndex as discussed before. In addition, we design BFSMinHash, which combines Bloom filters and MinHash algorithm, to further reduce the size of

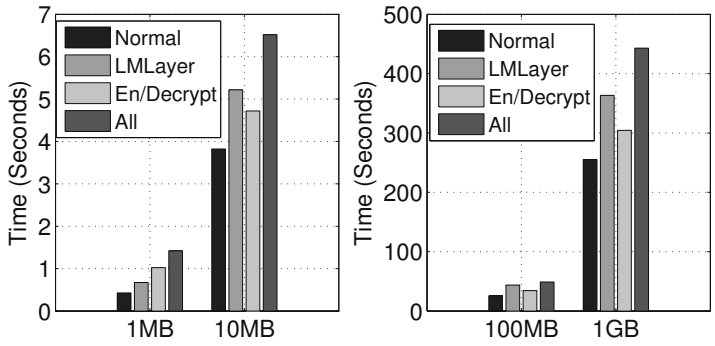

Figure 4.11: Time cost with different configurations by varying file size

similarity-preserving fingerprints by 1/16. In StoreSim, there are four main components in the client: deduplication based on SHA-1 signature, LMLayer based on BFSMinhash and SPClustering, encryption/decryption based on AES-256 (same with that employed by SpiderOak[113]) and bundling based on ZIP. We evaluate the overhead introduced by LMLayer in terms of four configurations: 1) *Normal* : deduplication and bundling; 2) *LMLayer* : deduplication, LMLayer and bundling; 3) *En/Decrypt*: deduplication, encryption/decryption and bundling; 4) *All*: all together. We compare the time cost by varying the size of files from 1MB to 1GB and Figure 4.11 shows the results. The time cost starts from dividing input files into small chunks and ends with assembling chunks to the original file. The *En/Decrypt* mode has an additional overhead since it has to decrypt the chunks before assembling. We discover that for small files of size less than 10MB, the overhead introduced by LMLayer is almost the same as the *En/Decrypt* mode. Especially in the case of 1MB, the performance of *LMLayer* is better than that of *En/Decrypt* mode. We conjecture this is because compared to *En/Decrypt* mode which needs key setup, there is no initialization overhead for measuring information leakage. For the large files (both 100MB and 1GB), the overhead of *LMLayer* is about 20% higher than that of *En/Decrypt*. In all cases, we notice that even in the *All* mode with all components running, the time cost is still tolerable for cloud storage services.

Syntactic vs Semantic. In our work, the information leakage function is designed based on syntactic similarity metric rather than semantic measures such as semantic similarity, semantic relatedness and semantic distance [114]. Thus, our system is incapable of detecting the private data such as financial documents and compromising photos in a semantic manner. Distributing data based on semantic measures is an orthogonal task to ours, since efficiently analyzing semantic similarity in users' data involves data curation and sophisticated machine learning techniques, which are not always time efficient for large datasets. Our future work will focus on developing efficient algorithms of optimizing privacy in multicloud storage based on semantics. These algorithms can be incorporated in our StoreSim as plugin modules.

Encryption vs StoreSim. Encryption is the most strongest and effective way to prevent information leakage. However, if all the data are encrypted, users will not be able to enjoy many other services provided by storage service providers such as file sharing or collaboration. Most of these services cannot operate over the ciphertext. Though there are some works on homomorphic encryption (i.e., allows computations to be carried out on ciphertext) [115], resorting to encryption as the ultimate solution to

prevent information leakage requires rewriting most of the cloud storage applications, which is evidently non-realistic. StoreSim provides an alternative approach to reducing information leakage to each CSP by optimizing data chunks distribution algorithms. In addition, StoreSim can also incorporate encryption after detecting near duplicate chunks and placing them together. Then, the information leakage can be reduced even if the encryption key is exposed. As shown in the Section 4.6.5, there is no single point of attack which can leak the wholesale information.

4.7 Related Work

Untrusted storage cloud: Depot [116] and SPORC [117] assumed that the storage clouds are unstrusted and fault-prone black boxes. However, both their work employed only a single cloud which has both compute and storage capacity. Our work is different since we consider a mutlicloud in which each storage cloud is only served as storage without the ability to compute. The earlier previous work such as Cooperative File System (CFS) [118] and Samsara [119] designed their storage system with a peer-to-peer network comprised of potentially untrusted nodes. Our work targets to use storage cloud without using decentralized P2P protocol and optimize the data chunk placement in a centralized way.

Multicloud storage services. Our work is not alone in storing data with the adoption of multiple CSPs, e.g., SPANStore [22], DepSky [3] and NCCloud[30]. However, these work focused on different issues such as cost optimization [22], data consistency and availability [3] and service response time [38]. Other efforts[120] on the cloud orchestration provided deployment plans in terms of the tradeoff between price and performance. Unlike these works, our work focuses on the information leakage optimization for storage service in a multicloud environment by exploiting information similarity caused by the synchronization of modified data. Supplementary efforts on overcoming vendor lock-in, DepSky [3] minimized the cost of data transfer from one cloud to another by storing only a fraction of the total amount of data in each cloud while Scalia [31] employed the data replication at a higher storage cost. However, in StoreSim, we provide a user-specific weight for each cloud which not only coordinates the fraction of storage load for each cloud but also prevents the information leakage across the CSPs. Other studies have focused on measurement analysis of cloud storage services [98, 99]. Their work provided us with many insights on designing and implementing StoreSim. But their work failed to reveal the optimization aspects of information leakages of the commercial CSPs they studied.

Cloud security. Many studies [2, 121, 122] focus on security and privacy aspects which are major obstacles of cloud adoption for both individuals and companies. Previous work [2] proposed a semantic framework based on crowd-sourcing to determine the sensitivity of items and diverse attitudes of users towards privacy. Bohli *et al.* [121] provided a survey for four different multicloud architectures with various security and privacy-enhancing designs. The architecture of StoreSim is one of them, which allows distributing fine-grained fragments of the data to distinct clouds. Our work further implements the StoreSim system with new information leakage measures.

Near-duplicate detection. Li *et al.* [123] proposed a privacy loss measure based on the JS-divergence distance which is a method of measuring the similarity between two probability distributions. Inspired by their work, we design our information leakage function based on similarity. To compute the information leakage, we need to compute the pairwise similarities. MinHash [104, 106] and SimHash [104, 111] were designed for detecting the near-duplicate web pages based on Jaccard and Hamming distance, respectively. However, their work cannot apply to our work directly due to heavy computation and

high storage overhead. To the best of our knowledge, this is the first work which applies near-duplicate techniques for preventing information leakage in multicloud storage services.

4.8 Conclusion

Distributing data on multiple clouds provides users with a certain degree of information leakage control in that no single cloud provider is privy to all the user's data. However, unplanned distribution of data chunks can lead to avoidable information leakage. In this chapter, we presented StoreSim, an information leakage aware storage system, to optimize the information leakage in the multicloud environment. StoreSim achieves this goal by using novel algorithms, BFSMinHash and SPClustering, which place the data with minimal information leakage (based on similarity) on the same cloud. Through an extensive evaluation based on two real datasets, we demonstrate that StoreSim is both effective and efficient (in terms of time and storage space) in minimizing information leakage during the process of synchronization in multicloud. Furthermore, through our attackability analysis, we show that StoreSim not only reduces the risk of wholesale information leakage but also makes attacks on retail information much more complex.

Part IV

Sharing

Chapter 5

CoShare: A Cost-effective Data Sharing System for Data Center Networks

Great undertakings have small beginnings , and difficult tasks are tackled from where it's easy

Laozi, Dao De Jing

5.1 Introduction

In the era of Big data, heterogeneous data are generated at a huge scale from many different domains, such as online social networks (OSNs), sensor networks, medical images, scientific measurements (e.g., CERN [58]) and Internet-wide measurements. This brings new challenges on how efficiently share these big data for research community and industry. Hereafter, we consider two typical examples of applications that highlight big data sharing scenarios.

Example 1: The social data generated from online social networks such as Twitter, Facebook and FourSquare is rich in semantic content on almost every aspect of human life, which has a broad appeal for academia, industry and governments. Due to the openness of social network platforms, researchers can obtain the social data through Open APIs to verify their hypotheses, to mine the interesting patterns, and to develop business models for real applications. Invariably different organizations undertake such efforts in isolation. The generated datasets, ranging from hundreds of Megabytes to several Terabytes, are stored in local data centers and isolated, resulting in many small *data islands*. The value of these small data islands. e.g., in data analytics, is also limited, in that they only serve the immediate purpose of data collectors. If the data collected by different organizations is shared together, the community at large can have access to a larger data source allowing them to not only validate their hypotheses but also expedite wider collaboration and foster scientific progress.

Example 2: Similar to peer-produced systems like Youtube or Wikipedia, consider a collaborative sensor network system (e.g., SenseWeb in Microsoft [124]) to enable peer production of sensing applications. The contributors in the sensor network system deploy their own sensors or sensor networks for their own dedicated applications, such as environmental monitoring system for air pollution, surveillance camera network for security, heart rate monitor and step counter for runners, road safety for Intelligent

Transportation Systems (ITS), etc. All these data are isolatedly stored in many Small Data Centers (SDCs) in diverse geographical areas. However, if these data can be efficiently shared together, many more applications can be developed. Thus, we need an effective data sharing system to coordinate data transfers among different applications and collaborators.

However, this data sharing process among different data islands incurs a large amount of data traffic as well as a huge cost on network bandwidth. These two main issues may discourage individual data centers from sharing their data. This process becomes more challenging because of the lack of a cost-effective data sharing system which is able to coordinate the data transfer. A client-server approach like Content Distribution Networks (CDNs) [65] for data sharing, requires servers that have high storage capacity and bandwidth to maintain a centralized repository. On the other hand, sharing data in a P2P manner can distribute the high cost of ownership to different peers [125]. Nevertheless, current implementations of P2P data sharing systems such as BitTorrent, focus on fast data dissemination based on incentive compatible data sharing strategies, e.g., Tit for Tat. Their main aim is to maximize the utilization of peer's bandwidth without considering the cost of data transfers [126].

What is needed is an approach that allows individual data centers to collaborate in a lightweight manner while providing cost and performance-effective data transfers, i.e., the system should find the most cost-effective solution given the performance requirements. Traditional P2P file swarming solutions [126] are lightweight, but delegate upload duties to all involved nodes. In a system that seeks to transfer massive amounts of data[1] between data centers ideally only those nodes that are less costly should be selected for upload duties. Furthermore, the scheme should respect users' requirements in terms of data transfer time. For example, if multiple data centers based in Switzerland, Poland and Tunisia are sharing data, and if the dataset to be replicated does not require to be shared urgently, then the most relevant solution would be choose data centers based in Tunisia as the uploading nodes (assuming cheap bandwidth cost is in Tunisia). On the other hand, if the data is required to be replicated crucially in the network, then data centers in Switzerland and Poland should also be included among the uploading nodes.

Motivated by this problem, we design a **Co**st-effective data **Shar**ing system, **CoShare**, which allows users to specify these high-level performance goals. As an example, for a data sharing task, users can attach a priority such as *high, medium, low* to the task.The priority indicates their preference on the data sharing such as completion time and monetary cost. The system will map the user's requirements on data sharing tasks into resource requirements and determine *which* and *how many* resources need to be assigned to the tasks. Existing systems [61, 62, 125] lack efficient mechanisms for translating users' requirements into resource requirements. CoShare provides such a mechanism for a data sharing network of collaborating data centers. The key idea of CoShare is to efficiently manage the tradeoff between cost and performance in such a network. Specifically, our contributions can be summarized as follows:

- We analyze the tradeoff curve between time and cost in a data sharing network based on different sharing plans and present a system interface that allows users to specify their requirements.
- We formulate the problem of finding a sharing plan with both cost and performance constraints and present our algorithm to find a sharing plan that matches user's preferences.

[1]Which is of higher orders of magnitude in size than the smaller sized software updates and media files, usually served by P2P systems

- Through extensive experiments based on our simulator, we demonstrate that CoShare is able to find the desirable tradeoffs between cost and performance given varying user requirements and request arrival rates.

5.2 Data Sharing Networks

In this chapter, we consider a data sharing network of small data centers (SDCs) which represent local SDCs of different research organizations and companies [127]. Each SDC generates a dataset and then replicates this dataset to SDCs that already expressed their interest to it. This sharing process continues until all interested SDCs receive this dataset. In the following, we briefly present two typical network architectures for data sharing and discuss the tradeoff between conflicting cost metrics in a data sharing network.

5.2.1 Network Architecture for Data Sharing

Dropbox [11] is a representative of commercial cloud-based solutions that provide users with file synchronization services. As it has a centralized architecture, all the data needs to be uploaded to the cloud first and then other interested clients can download the data from the cloud. This client-server approach for data sharing requires dedicated servers and high network bandwidth, which leads to high cost for data sharing. This architecture is not suitable for our scenario since participant SDCs are self-organized, and with limited budgets for data sharing. On the other hand, P2P-based data sharing approaches distribute the large volume of data between peers directly without a central entity and thus there is no up-front cost on building the network. Nonetheless, current P2P swarm protocols such as BitTorrent and Avalanche fail to provide guarantees on the cost and performance of data sharing [125].

In this chapter, we design CoShare inspired by P2P network architecture. To manage the tradeoffs between performance and cost in the data sharing network, we add a centralized entity called *sharing manager* which coordinates the data transfer among SDCs[2]. The coordination is performed in terms of the *sharing mode* which determines the direction of data exchange. There are three types of sharing modes between SDCs: 1) *sharing*: SDCs can upload/download from each other; 2) *downloader*: download only; and 3) *uploader*: upload only. The communication between the sharing manager and SDCs belonging to the data sharing network can be summarized as follows. After receiving the sharing request from a given SDC, the sharing manager will ask each interested SDC to report its current bandwidth usage and generate a *sharing plan* considering both performance and cost of data sharing. Here, the sharing plan is defined as a set of sharing modes corresponding to each SDC in the network. The sharing plan will be sent to all SDCs. Once it is received by SDCs, they can communicate with each other directly by using BitTorrent-like protocols. The shared dataset is divided into small data blocks and then indexed by block hashes. After establishing connections, interested SDCs will exchange their indexes and then synchronize the dataset by requesting missing blocks from others.

[2]We note that this is a lightweight component and need not be centralized. If nodes use gossip protocols or Distributed Hash Table (DHT) services to keep abreast of the state of the network, and the sharing node runs a lightweight tracker, then no central entity is needed.

5.2.2 Cost Metrics in Data Sharing

The goodness of a data sharing network can be measured in terms of several metrics such as monetary cost, link utilization, network congestion, etc. In this chapter, we discuss two main metrics: *monetary cost* and *completion time*.

	AWS			**AZure**			**Google**			
Location \Usage	1GB	10TB	50TB	5GB	10TB	50TB	1TB	10TB	10TB+	**Destination**
US/Europe	0	0.09	0.085	0	0.087	0.083	0.23	0.22	0.20	China
Asia (Tokyo)	0	0.14	0.135	0	0.138	0.135	0.19	0.18	0.10	Austraila
Sao Paulo	0	0.25	0.23	0	0.181	0.175	0.12	0.11	0.08	Others

Table 5.1: Price models of three cloud providers (dollars per GB)

We define the *monetary cost* of serving a sharing request as the total bandwidth cost of all SDCs that contribute to the data sharing. Greenberg *et al.* [128] have revealed that the network cost amounts to around 15% of operational costs to a data center. The bandwidth cost is usually charged based on the percentile-based charging scheme. For example, the common billing method is referred to as the "95th Percentile Rule" [58] in which an Internet Service Provider (ISP) measures the traffic volume that a data center generates every 5 minutes. At the end of a charging period, all 5-minute samples will be sorted and the highest 5% samples will be discarded. However, when it comes to the charging scheme between data center and users, the 100-percentile charging scheme is applied. This implies that data center providers will charge all the data traffic without removing the peak traffic. Table 5.1 summarizes the piece-wise linear price models for network bandwidth of three cloud data centers, namely Amazon, Google and Azure. For instance, for AWS data center in Asia, the price of data traffic within 1GB is free while that of traffic between 1GB and 10TB is charged at 0.14$/GB. Table 5.1 clearly shows that besides the total volume of data traffic, geographic location is also a crucial factor that influences the bandwidth cost. For example, the price of bandwidth in Sao Paulo is nearly two times than that in US/Europe area in Amazon data center. In particular, the price of bandwidth in Google also depends on the location of data traffic destination regardless of source location. Furthermore, it is worth to mention that all the cloud data centers only charge on upload links while downloading from external Internet is usually free. Thus, in our scenario, we assume that the participating data centers are located in different areas with various bandwidth price models. This setting gives us the opportunity to optimize the total cost of data sharing.

The other metric we consider is the *completion time* of serving a sharing request. For each sharing request issued by a SDC, the completion time is measured from the time that the source SDC starts to upload the dataset until the time that all interested SDCs finish downloading all the dataset. Many factors have an impact on the completion time. This includes the dataset size, bandwidth and monetary cost constraints.

5.2.3 Tradeoff in Cost Metrics

As we mentioned in Section 1, there exists a tradeoff between cost and performance (in terms of transfer time) in a data sharing process. For better understanding, consider the scenario depicted in Figure 5.1 where we have a swarm of four SDCs with the same bandwidth capacity but different prices. The SDC 1 wants to share a dataset with the other three SDCs. To achieve this goal, different plans can be employed as is shown in Figure 5.1. Specifically, Figure 1 (a) shows that in the Client-Server plan (P1), three

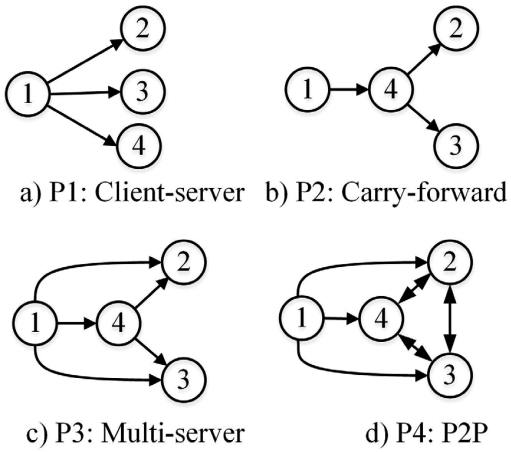

Figure 5.1: Different sharing plans

client SDCs (with *downloader* modes) only download the data from the server SDC 1 (with the *sharing* mode) while Figure 1 (b) depicts the Carry-Forward plan (P2) according to which SDC 1 shares the dataset with SDC 4 first and then SDC 2 and SDC 3 download the data from SDC 4. The benefit of P2 is that it can reduce the cost if the price of SDC 4 is lower than that of SDC 1. As we assume that the bandwidth capacity is the same, the completion time of plans P1 and P2 is also the same. Without the loss in completion time, the cost of P2 is lower than that of P1. In this sense, we say that P1 is *dominated* by P2. The plan P2 can even be the best plan in terms of cost if we assume that the price of SDC 4 is the lowest among all four SDCs. On the other hand, to reduce the completion time, we can add more upload capacities to transfer the data. This is depicted in Figure 5.1 (c) which shows the Multi-Server scenario (P3) in which two more links are added compared to P2 to accelerate the data sharing. Thus, SDC 2 and SDC 3 can download from SDC 1 and SDC 4 simultaneously. We can observe that P3 is better with respect to the completion time, whereas P2 is better in terms of cost. In this case, we say that P2 and P3 are *non-dominated* sharing plans, i.e., there is no single plan better than the other in both cost and performance. Figure 5.1 (d) depicts the P2P sharing plan (P4), in which the best completion time can be achieved since all SDCs' bandwidth are fully used; however this makes it costlier.

From Figure 5.1, we can see that each sharing plan represents a tradeoff between performance and cost. Given a sharing plan, we can plot its cost and performance on a 2-dimensional plane. In Figure 5.2, the four sharing plans with different cost and performance are presented. Given a target performance on the completion time, we define a non-dominated sharing plan as the one with *the minimal cost* for that target performance. We refer to the curve in Figure 5.2 as the '*tradeoff curve*', which is comprised of all the possible non-dominated sharing plans. In Figure 5.2, P1 is not on the tradeoff curve because the cost of P1 is not the minimal one given the completion time *t5*. P2 has the same completion time at a lower cost, and thus dominates P1. In this sense, only non-dominated plans are of interest. Selecting a sharing plan from all non-dominated plans largely depends on user's preferences. In CoShare, we design three priority levels, namely *high*, *medium* and *low*, for users to specify their preferences in order to manage their tradeoff on the data sharing. The priority indicates users' preference on the performance metric, i.e., completion time in our system. For example, if a user assigns high priority to a sharing request, CoShare will choose a non-dominated sharing plan with relatively faster completion time.

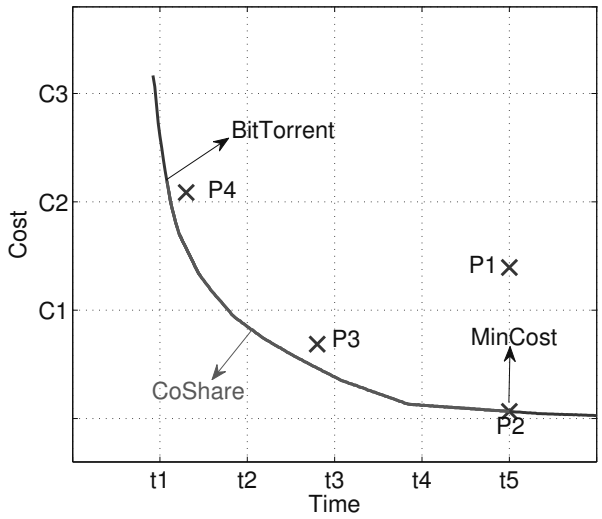

Figure 5.2: Tradeoff management by CoShare in terms of cost and time. While BitTorrent and MinCost are two extremes on the tradeoff curve (e.g., the most fastest but expensive versus the slowest but cheapest), CoShare aims at finding a desirable tradeoff and adapting to bandwidth usage.

Thus, the key questions that we aim to answer can be reformulated as follows: *Given a swarm of SDCs with different bandwidth capacities and price models, 1) how can we find all the non-dominated sharing plans? 2) From these plans, which is the desirable sharing plan that satisfies the user's requirements?*

5.3 CoShare : Cost-effective Data Sharing

In this section, we present the design of CoShare. Firstly, we discuss how to generate all non-dominated sharing plans on the tradeoff curve. Then, we will evaluate the goodness of sharing plans from the tradeoff curves in terms of users' preferences. Finally, we will present the algorithm in CoShare that selects the desirable sharing plan.

5.3.1 Generating Non-dominated Sharing Plans

We consider a data sharing network composed of a set $\mathcal{P}$ of heterogeneous SDCs, which is denoted by $\mathcal{P} = \{p_1, \ldots, p_n\}$. Each SDC $p_i \in \mathcal{P}$, is characterized by its maximum upload and download capacities, denoted respectively by u_i and d_i. We adopt the 100-percentile price model with free charge on ingress traffic, i.e., the data center only charges on the egress links. Thus, the cost function can be defined as a piecewise linear nondecreasing function $\mathcal{C}(x)$, where x indicates the total volume of data. A sharing task is specified as a tuple $\{p_{src}, \mathcal{S}(p_{src}), Q\}$ where p_{src} is the SDC of data source, $\mathcal{S}(p_{src}) = \{p_j | p_j \in \mathcal{P} \wedge j \neq src\}$ is the set of SDCs which subscribe to the dataset from the p_{src} and Q is the size of the dataset. The data sharing process follows the one-to-many data transfer model, i.e., dataset is replicated from one SDC to the interested SDCs in the network. Suppose $p_s \in \mathcal{P}$ proposes a

sharing task $\{p_s, \mathcal{S}(p_s), Q\}$. Given the current available bandwidth ratio ρ, we formulate the optimization problem to minimize the total sharing cost as:

$$min \sum_{i=1}^{n} \mathcal{C}_i(\sum_j x_{ij}) \tag{5.1}$$

subject to

$$\sum_i \sum_j x_{ij} = (|\mathcal{S}(p_s)| - 1) * Q \tag{5.2}$$

$$0 \leq \sum_{j \neq i} x_{ij} \leq u_i * \rho * T, \forall i, j \tag{5.3}$$

$$\sum_{i \neq j} x_{ij} = Q \leq d_j * \rho * T, \forall i, j \tag{5.4}$$

$$Q \leq \sum_{j \neq s} x_{sj} \leq u_s * \rho * T \tag{5.5}$$

$$\sum_i x_{is} = 0, \forall i \tag{5.6}$$

$$x_{ij} \geq 0, \forall i, j, i \neq j \tag{5.7}$$

where x_{ij} is the total amount of data uploaded from p_i to p_j and T is the expected transfer performance in terms of completion time. As we mentioned in Section 5.2.2, we evaluate the performance of data sharing not only in terms of completion time but also in terms of the cost which refers to the monetary fees for bandwidth cost. The objective of problem 5.1 is to minimize the total cost with the performance constraints on completion time T. Constraint 2 guarantees that all interested SDCs have downloaded this data of size Q. Constraints 3 and 4 specify the bandwidth constraints for both upload and download capacities. With constraints 5 and 6, we ensure that the SDC of data source p_s uploads at least all the data without any data downloading. The parameter ρ indicates the mean available bandwidth ratio of the data sharing network, which reflects the current load in the network. Thus, given a target performance T, we can compute the optimized cost of data sharing by solving problem 5.1. By varying all possible values of T, we can derive the tradeoff curve. Considering this curve as an input, the remaining issue that we have to resolve is to determine the data sharing plan with desirable tradeoff that satisfies user's requirements.

5.3.2 Goodness of Sharing Plan

Through solving problem 5.1, we can obtain the tradeoff curve with all non-dominated sharing plans. No single plan on the tradeoff curve has both lower cost and lower completion time. Our system will evaluate the goodness of a sharing plan with respect to users' preferences. To map users' preferences to the sharing plan, we define an adjustable tradeoff factor tf as the *marginal utility* which indicates the marginal improvement in completion time by adding additional unit costs. For example, given two sharing plans sp_i, sp_j with different completion time and cost $< t_i, c_i >$ and $< t_j, c_j >$, the marginal utility of two plans is computed as absolute value of $tf = |(c_j - c_i)/(t_j - t_i)|$. In this sense, tf is the slope of two points, which evaluates the rate of change of sharing cost with respect to the sharing

completion time. Taking the tradeoff curve given by Figure 5.2 as an example, a sharing plan with tf in the curve is the point where the slope of the curve becomes higher than tf when going from right to left. tf equals to 0 indicating a sharing plan with the lowest cost while higher value of tf means less completion time at the higher cost. Based on this analysis, different priorities of sharing tasks correspond to different values of tf. We have to note that the value of tf is influenced by the system configurations, such as the total available bandwidth, request arrival rate and dataset size.

Algorithm 5.1: Sharing plan search algorithm

Require: a sharing request $req = \{p_s, \mathcal{S}(p_s), Q\}$, request priority $l \in \{high, medium, low\}$, request arrival rate λ
Ensure: a sharing plan $\mathbf{sp}$
1: $t_{start} = max\{Q/u_s, Q/min\{d_i\}\}$; //lower bound
2: $t_{exp} = meanReqInterArrival(\lambda)$; //upper bound
3: peerDCStatus = $getPeerDCStatus()$; //obtain each SDC status
4: **while** $t_{min} <= t_{max}$ **do**
5: $t_{mid} = midtime(t_{start}, t_{exp})$;
//apply linear programming solver
6: solverResult = $LPSolver(req, l, peerDCStatus, t_{mid})$;
7: **if** $solverResult.accept < 0$ **then**
8: $t_{start} = t_{mid} + 1$
9: **else if** $solverResult.accept > 0$ **then**
10: $t_{exp} = t_{mid} - 1$
11: **else**
12: break;
13: **end if**
14: **end while**
15: **return** solverResult.$\mathbf{sp}$

5.3.3 Searching Desirable Sharing Plan

To determine the tradeoff factor tf, we firstly identify two main factors in problem 1, dataset size Q and mean available bandwidth ratio ρ, which exert great influence on the generation of tradeoff curves. The exact values of these two factors can only be known at the time of sharing while other factors such as upload/download capacities and bandwidth cost can be derived when the SDC joins the network. Given the varying available bandwidth in the system over time, the same task may generate different tradeoff curves while on the other hand, tasks of different sizes may also present similar performance. For example, if the tradeoff factor tf is set as a fixed value, the first task of size Q_1 can achieve this tradeoff as long as the available bandwidth reaches $\rho < 1$ while the second task of the larger size $Q_2 > Q_1$ can reach the same tradeoff only when all the bandwidth is available, i.e., $\rho = 1$. Thus, if we fix the value of tf for all tasks, some tasks have to wait until there is enough available bandwidth to satisfy the user-specified tradeoff. To avoid this issue, we relax the tradeoff factor into an interval and map user's preference to the corresponding tradeoff interval. For instance, if the tradeoff factor interval is $[tf_1, tf_2]$, the task can be served as long as the system can search a sharing plan with the tradeoff factor $tf \in [tf_1, tf_2]$. The final intervals for different priorities are derived based on the whole system configurations. We will provide a comprehensive discussion regarding the tradeoff factor in Section 5.4.

To search a sharing plan that satisfies a desired tradeoff, we apply the binary search over the performance space, i.e., the space of completion time. Algorithm 6.2 shows the skeleton of sharing plan

searching algorithm. The bandwidth Constraints 3 and 4 in problem 1 show that the completion time may be constrained either by the upload bandwidth of data source or the minimal download bandwidth of other SDCs. In other words, the completion time of data sharing satisfies $t_{start} \geq max\{\frac{Q}{u_s}, \frac{Q}{min\{d_i\}}\}$. This gives the start value of the completion time (line 1 in Algorithm 6.2). On the other hand, we expect that the current sharing request can be served before the next request arrives in the system. Thus, the expected completion time t_{exp} can be the average inter-arrival time of requests (line 2 in Algorithm 6.2). Given start time t_{start} and expected completion time t_{exp}, we can carry out the binary search over the time interval $[t_{start}, t_{exp}]$ (lines 4-14 in Algorithm 6.2). For each search, the system will apply a linear programming solver to solve problem 1. In practice, t_{exp} can be extended to a *hard* deadline by a scale factor if we cannot seek for sharing plan in the interval $[t_{start}, t_{exp}]$. After extension, if there is still no sharing plan with desirable tradeoff, the sharing request will put back into the request buffer to wait for more available bandwidth.

5.4 Evaluation

In this section, we will introduce the evaluation methodology of our system by presenting the simulator, the workload as well as the performance metrics. Then, we will illustrate detailed experiments and results under different settings.

5.4.1 Simulator

We developed a discrete event simulator to simulate the interactions between entities forming the data sharing network. More specifically, two entities in CoShare are implemented: the sharing manager and SDCs. The sharing manager is responsible for coordinating the data sharing process among different SDCs. The incoming sharing requests generated by SDCs will be buffered in a queue and be processed by the sharing manager in a first-come-first-served manner. To serve the sharing request, the manager will firstly ask each interested SDC to report their current bandwidth usage. Based on this information, it searches for a sharing plan by solving problem 1. Without loss of generality, we assume the cost function to be linear, i.e., $c(x) = ax$ where x is total volume of egress traffic. To solve problem 1, the sharing manager employs MOSEK [129] to solve the linear programming problem and generate the tradeoff curve. After finding a desirable plan, it will send the sharing plan to each interested SDC in the network. Once the sharing plan is received, interested SDCs can start the data sharing process. From the network performance perspective, we assume that the bandwidth of each SDC is equally shared by all the links for both download and upload. Each SDC calculates the download/upload throughput as the ratio of the total size of data download/upload at a regular measurement interval, i.e., 100ms. As mentioned in Section 5.2.1, the data is transferred in a unit of small data block and the size of small block in our simulator is set to 1 MB. To compute the transfer delay, the system will first find out the minimal rate between local upload rate and remote download rate. Then, the transfer delay is calculated as the ratio between the size of a data block and this minimal rate.

5.4.2 Simulation Setup and Metrics

Setup: We simulate a swarm of 50 SDCs with the download and upload bandwidth uniformly distributed in the intervals [80Mbps, 160Mbps] and [40Mbps, 80Mbps] respectively. The price of bandwidth is obtained from public cloud providers and CDN providers. The most expensive price is 0.25 $/GB from Amazon data center in south America (Sao Paulo) while the cheapest one is 0.06 $/GB from MaxCDN in Europe area. The average price of all 50 SDCs is about 0.153 $/GB. The size of datasets for each sharing request is uniformly distributed in the interval [300MB, 500MB]. For each dataset, we randomly generate the number of interested SDCs which belongs to the interval [16, 32] (i.e., the average size of interested SDCs is about 24). Unless otherwise stated, the default priority of the sharing request is *low*. We simulate a total of 200 requests with a total aggregated data size of nearly 2TB. Requests are generated according to a Poisson distribution with a mean arrival rate of λ. We tune the system bandwidth utilization by varying λ. For each experiment, we run the simulator 10 times with different random seeds and the variance in results is as low as about 5%.

Metrics: The efficiency of CoShare is evaluated through two groups of performance metrics: 1) *Sharing cost and completion time*: Sharing cost is the monetary fees on the bandwidth and completion time is the total time consumed by serving a sharing request; 2) *Task throughput and price*: Since individual sharing requests are of different sizes, i.e., each task replicates different amount of data with interested SDCs, we compute the task throughput as the ratio between the total amount of data transferred and the completion time. The task price is calculated as the ratio between the total sharing cost of a task and the total amount of data transferred.

5.4.3 Simulation Results

Hereafter, we present the performance evaluation of CoShare. In particular, we are interested in answering the following questions: how to determine the value of tradeoff factors tf? What is the effect of priority levels specified by different users on the performance of tasks? What is the impact of the system workload on the average cost and aggregated throughput of the whole system? What are the benefits achieved by our design? To answer all these questions, we have conducted three groups of experiments.

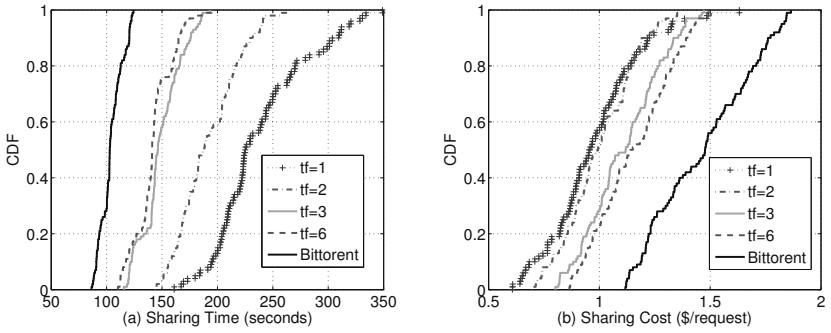

Figure 5.3: CDF of tasks with fixed tradeoff factors

Fixed Tradeoff Factors. In this group of experiments, all the tasks arrive with the same tradeoff factor tf. The request arrival rate is set to $\lambda = 12$ requests per hour, i.e., requests arrive about every 300 seconds. Under this configuration, we vary the tradeoff factor tf from 1 to 6. Figure 5.3 depicts

the variation of the Cumulative Distribution Function (CDF) of tasks as function of time and cost under different tradeoff factors values. Note that for comparison, we also simulate the BitTorrent protocol under which all SDCs contribute their bandwidth to finish data sharing as fast as possible. As we can clearly see, BitTorrent has the shortest completion time, but at the *avoidable* high cost compared with our system. This is explained by the fact that SDCs using BitTorrent protocol aim to finish downloading datasets as fast as possible without caring about the transfer cost. A closer look to Figure 5.3(a) shows that BitTorrent enforces that all SDCs finish data sharing within about 120 seconds. After this period, they stay idle for about another 180 seconds before serving the next request, which makes it meaningless to get fast speed at the high cost as shown in Figure 5.3(b). From other side, Figure 5.3(a) and 5.3(b) also illustrate that tasks with lower tradeoff factors are served with less bandwidth in order to reduce the sharing cost, thus leading to longer completion time. Specifically, three groups of tasks with $tf = 2, 3, 6$ can also finish the task before the next request arrives at the much lower cost (by reducing about 25% compared with BitTorrent's cost). The tasks with $tf = 4, 5$ are omitted in the Figure 5.3 since their performance is between $tf = 3$ and $tf = 6$ and highly similar with $tf = 6$. The reason for the similar performance of tasks with different tradeoff values is that the tradeoff curve decreases slowly in this interval. For example, as it is shown in Figure 5.2 (cf. Section 5.2.3), the sharing plans on the tail of the curve (e.g., time between *t3* and *t4*) shall present similar performance since the curve decreases slowly. Tasks with $tf = 1$ have the lowest cost reduced by about 20% than that of group with $tf = 3$ and by about 50% than that of BitTorrent. However, from Figure 5.3(a), we observe that only 85% of them can finish before the next request arrival, which means that about 15% requests will be scheduled on the SDCs with more expensive cost. As a result, we can notice that about 15% of requests with $tf = 1$ has much higher cost than others as it is reported by Figure 5.3(b).

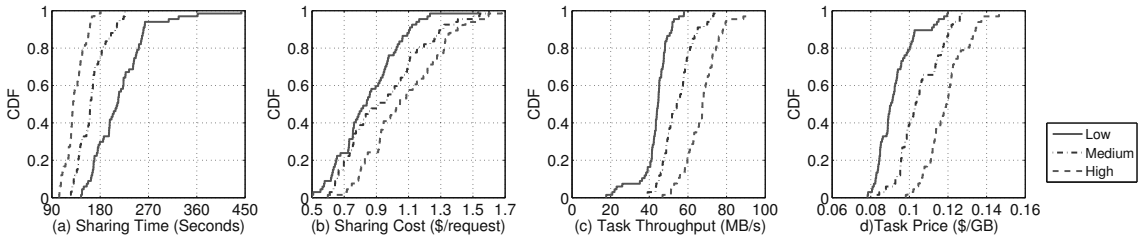

Figure 5.4: Tradeoffs for tasks with high, medium and low priority.

User-specified Priority. Through this set of experiments, we aim to study the effect of the user-specified priorities on the task performance. In this scenario, all the tasks arrive in CoShare with different user-specified priorities. Based on the observations deduced from Figure 5.3, we map these priorities, i.e., high, medium, low to the tradeoff intervals [3,6), [2,3) and [1,2), respectively. One criteria for selecting suitable intervals is to find the turning points in the tradeoff curve where the curve decreases quickly. Other criteria such as task price and throughput can also be applied to find the interval that can differentiate sharing plans in terms of distinctive user-specified priorities. Then, we assign three priorities to tasks with the same proportion (i.e., nearly 33% for each priority). Figure 5.4 demonstrates the tradeoff of tasks with different priorities with respect to different metrics. It can be clearly seen that CoShare is able to apply users' preferable tradeoff into data sharing effectively. Specifically, for sharing completion time and cost depicted by Figure 5.4 (a) and 5.4(b), all tasks with high priorities can finish within around

180 seconds at the high cost (i.e., about 60% of the tasks with costs large than 1.0 dollars) while only 25% of tasks with low priorities can achieve completion within approximately 180 seconds with low sharing costs (i.e, 80% of them have costs less than 1.0 dollars). Furthermore, Figure 5.4(c) and Figure 5.4(d) present respectively the CDF of task throughput and price. We can observe that the average throughputs of tasks with high, medium, low priority are about 42, 50 and 65 MB/s at the price of about 0.093, 0.106 and 0.120 $/GB, respectively. Thus, based on these results, we can safely conclude that CoShare provides the *guarantee* on cost and performance for serving users' sharing requests. In particular, this guarantee is tunable by giving different values or intervals for tradeoff factors. For example, in order to increase/decrease the price of high priority task, we only need to adjust the corresponding tradeoff interval.

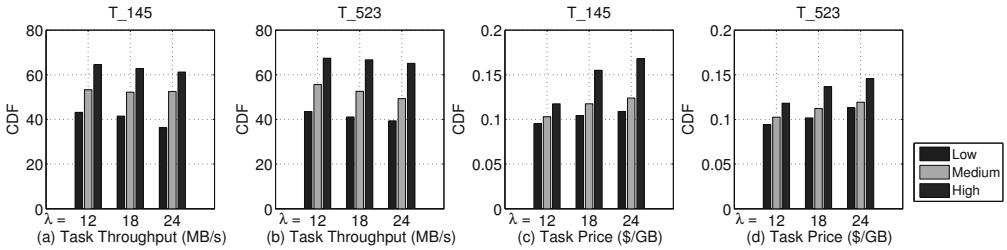

Figure 5.5: Tasks with priorities in different proportions under various request arrival rates.

Effect of request arrival rate. Our objective through this group of experiments is to investigate the impact of the request arrival rate on both throughput and price of tasks with different priorities. For this purpose, we generate two groups of tasks with various priorities in different proportions. In the first group, denoted by T_145, only 10% of generated tasks have a high priority while tasks with medium and low priorities represent respectively 40% and 50% of the total tasks. In the the second group, T_523, the ratio of tasks with high, medium and low priorities is fixed respectively to 50%, 20% and 30%. The difference between these two task groups is that the majority of SDCs in T_145 prefer to share the data at the low cost while those in T_523 are weighted in favor of sharing it in a fast way. We run the experiments by varying the request arrival rate $\lambda = 12, 18, 24$ requests per hour. There are two main findings that can be observed from Figure 5.5. First, SDC's assignments of different priorities in sharing requests exert little impact on the system performance with regard to the task throughput. As it is shown in Figure 5.5(a) and 5.5(b), we observe that task throughput of two groups of tasks is similar. Moreover, the system can also guarantee enough throughput for sharing requests of each priority under different request arrival rates. Second, keeping the same experimental settings, Figure 5.5(c) and 5.5(d) depict the variation of the price of both groups of tasks. As expected, we observe that the task price is greatly affected by the request arrival rates. Not surprisingly, we can observe that the task price increases with the increase of request arrival rate. This can be explained by the fact that more expensive bandwidth is utilized as more requests arrive at the system within the same time unit. We also observe that in Figure 5.5(c) where the most of tasks are with low priority, the variation in the task price of different priorities are much higher than these in Figure 5.5(d) where the majority of tasks have high priority. In the group of tasks T_523, and under the highest request arrival rate, most of the bandwidth resources with lower price are occupied by requests with high priorities. Under this situation, tasks with low priority are served by expensive bandwidth with high probability, which results in relatively higher task price. Thus, the difference in the

task price of various type is diminishing. This also illustrates that the focus of CoShare is the tradeoff between the time and cost rather than minimizing the cost only. For each request, CoShare will find a sharing plan satisfying the corresponding tradeoff factor rather than the one with the lowest cost.

Discussion. Based on our extensive simulation of data sharing networks, we show that CoShare effectively helps users to manage the tradeoff between cost and performance with respect to three different priority levels. However, we may find that the advantage gained by CoShare will be decreased with the increase of the number of requests with high priorities. Table 5.2 summarizes the cost-savings of two groups of tasks under different request arrival rates compared with the cost of BitTorrent. We can observe that the relative cost-savings achieved by CoShare can decrease from 34% to 10.8% as more requests with high priority arrive at the highest request arrival rate ($\lambda = 24$).

	$\lambda = 12$	$\lambda = 18$	$\lambda = 24$
T145	34.0%	18.6%	11.6%
T523	29.1%	18.3%	10.8%

Table 5.2: Total cost savings of CoShare compared with BitTorrent's cost.

On the other hand, we also observed that determining suitable tradeoff factors for a data sharing system is an iterative process. This suggests that the system designer will have to experiment in order to find the suitable tradeoff factors for her particular data sharing network. In the real network scenario, this process could take a very long time, which may incur time and monetary costs. Thus, our methodology provides an easy and economical way for system designers to speed up this process.

Finally, we discuss three major impediments that can limit the practicality of our model. Firstly, do SDCs, especially those that belong to areas of low bandwidth cost, have any interest on our globally optimized solution? Our global optimization can obtain the optimal cost-savings *overall*. The problem of how to distribute these cost-savings to individual members such that no single SDC has the incentive to leave the swarm is an orthogonal issue to ours. Secondly, do these SDCs report their bandwidth cost honestly? One way to solve this issue is to employ a trust model [130] for our data sharing system to discourage SDCs' dishonest behavior. Thirdly, since most researchers commonly store their data locally, can we extend our model to a more general setting that incorporates, besides SDCs, local hard disks? Currently, we only focus on the data center networks in which the number of peers is far smaller than that in traditional P2P network. Thus, optimization in a relatively small SDC network can be easily solved by using existing linear programming packages such as MOSEK. However, considering individual researchers with local hard disks will lead to a dramatic increase in the number of peers, which will in turn will increase the number of variables quadratically. When the number of variables is too large, no generic linear programming solver that we are aware of can be applied. In that case, we will have to develop a specific approximate algorithm to solve Problem 1 presented in Section 5.3, e.g., a greedy algorithm. Another solution to include local disks of individual researchers is to assign the nearest SDC to each peer. For each instance of data sharing, peers can upload their data to the nearest SDC first and then the data will be replicated from this nearest SDC to the others using the CoShare system directly.

5.5 Related Work

Many related works focused on scheduling data transfers among data centers with different optimization objectives. The store-and-forward strategy adopted by [36, 58, 59, 60] aims to schedule the data transfer in order to minimize the cost [36, 58] and to readjust to the resource fluctuations [59, 60]. To increase the throughput and thus minimize the transfer time, StorkCloud [61] integrated multi-protocol transfers aiming at optimizing the end-to-end throughput while considering the link capacity, disk rate and the CPU capacity. NetStitchter [59], employed a network of storage nodes to schedule the data transfer in a store-and-forward manner based on predictions on the available leftover bandwidth at access and backbone links. In order to decrease the cost, Postcard [36] formulated an online optimization problem to minimize costs on inter-datacenter traffic by exploiting the price discrepancy in different locations. Other works such as [62] designed a set of transfer strategies to readjust to different network conditions (e.g., network latency, available bandwidth) between cloud data centers whereas [63] focused on the evaluation of different data transfer protocols for big data. However, none of these works considers the tradeoff between the time and the cost during the data transfer process. Unlike previous presented works, our work manages this tradeoff through system interfaces, which allows users to specify their preferences, and provides an efficient mechanism to find this tradeoff. Though Wu *et al.* [43] proposed scheduling approaches for bulk data transfers with different urgency levels. In their work, the priority assigned to a data transfer only concerned the job scheduling without taking into account the time-cost tradeoff.

Our work is not alone in managing the tradeoff between time and cost. Works presented in [64] and [65] optimized the time and cost performance for online services (e.g., search engine) and content multi-homing (e.g., CDNs), respectively whereas our work focused on the data sharing networks. Tudoran *et.al* [66] proposed transfer as a service for multi-site cloud with considering the cost in both computation and network. They did not consider the tradeoff between the cost and time. Moreover, they also failed to provide interfaces for users requirements. As for data sharing in P2P networks, most of the works focus on maximizing the throughput [125, 131, 132] using the BitTorrent protocol. Capota *et al.* [131] formulated a maximum flow optimization problem while [132] improved neighbor selection based on traffic locality. Peterson *et al.*[125] presented the response curve of P2P swarm, which represents the swarm bandwidth as a function of seeder bandwidth. Their system assigns the seeder resources to the swarms with the steepest response curves. However, their optimization still did not consider the bandwidth cost. To the best of our knowledge, ours is the first work that manages the time-cost tradeoff for data sharing networks.

5.6 Conclusion

This work serves as a first step towards studying the tradeoff between cost and performance for massive data sharing among small data centers. We proposed CoShare, a cost-effective data sharing system that maps users requirements into resource requirements for managing the time-cost tradeoff. Through extensive experiments, we demonstrated that CoShare is effective in this mapping and guarantees the desirable cost and performance tradeoff for the data sharing process. Future efforts will focus on implementing a real data sharing system based on CoShare. We would also like to incorporate a semantic component for data sharing. For instance, this component can be able to manage the data deduplication

in transferred datasets (e.g., Twitter datasets), which further reduces the total amount of data that have to be transferred.

Chapter 6

Data Summarization with Social Contexts

Therefore, since brevity is the soul of wit, and tediousness the limbs and outward flourishes, I will be brief.

William Shakespeare, Hamlet

6.1 Introduction

Enormous amounts of social data that are generated on sites such as Twitter, Facebook, Yelp, hold great value for both researchers and practitioners in understanding social behaviors. For instance, social data has been applied on various applications such as social event detection and sentiment analysis. Though social media data provides great opportunities, it also brings about many challenges due to its sheer size. Firstly, it is very expensive to store, share and process data of such large scale [133]. Secondly, it is computationally expensive to build analytical models to power social services and applications. Thirdly, more data does not necessarily mean more useful data [134]. Sometimes additional training data could result in worse performance [135], and it is challenging to identify those data that are truly useful.

Data summarization [69, 70, 73] has shown its effectiveness in preparing large-scale data for data analytics. Different from data compression (that reduces cost of storage and communication by compressing dataset into smaller size) or data sampling (that reduces the cost of training analytical models by randomly selecting a subset of data samples), data summarization aims to maintain (or improve) the performance of applications by selecting a subset of data that is considered to be useful. Generally, data summarization is done by transforming the problem into selecting a subset of data instances, with an objective function that quantifies properties of the selected subset. These properties could be representativeness [69, 70], diversity [71], informativeness [72], etc., which are defined in the context of different applications. However, existing methods design the objectives of data summarization purely based on attribute-value contents, ignoring social contexts which could provide valuable information. We take Twitter as a case study. Social contexts can refer to demographical information about users (e.g., age, gender, location), social status (e.g., number of followers) and their actions (e.g., reply, retweet, favorite, block). Intuitively, it can be conjectured that incorporating those tweets which are posted by influential users, or those that are retweeted by more users, into the summarized dataset, could be more beneficial

to a learning task. Thus, we are motivated to investigate how social contexts could be utilized for social data summarization.

In addition, topical information plays an essential role in many text mining tasks. Despite many work on summarizing social data in different aspects such as sentiments [136] or events [137, 138], very little work has yet focused on topic preserving data summarization. Therefore, our aim is to summarize social media data into a small subset while preserving topics in the original dataset. To this end, we analyze statistical topic models [139] and present three challenges in topic-preserving summarization for social media data. The first challenge is that a topic is a *latent* concept that needs to be learned from the dataset. Most of the existing work [69, 70, 71] in data summarization have explicit objectives with known parameters while, because of latent topics, our work needs to estimate unknown parameters based on the dataset. The second challenge is that even if we fix parameters in the objective function, we would still have to search over an exponential number of possible subsets to find the optimal subset. The last challenge is that the subset obtained from data summarization should maintain (or even improve) the performance of applications. Put bluntly, the performance of a trending topic application should not be much worse when it works on a summarized dataset as compared to its performance on the original dataset.

To address these challenges, we explore the use of social contexts in social media data to help topic-preserving summarization. Specifically, we study this problem using two real-world Twitter datasets based on two different applications, namely *Topic Discovery* and *Tweet Classification*. In doing so, we are confronted with the following questions: how to design an objective function for preserving latent topics in Twitter dataset? Is the social context in Twitter datasets helpful for topic-preserving summarization? Does the performance of applications on summarized subsets degrade as compared to their performance on the original datasets ? In addressing the above questions, this chapter makes the following contributions:

- Through analyzing statistical topic models, we present an objective function for preserving topics in Twitter data. We also present two challenges related to parameter estimation and the size of the search space for this function. To solve these two challenges, we first design a submodular model, called *E-model*, based on information entropy. Apart from solving the parameter estimation challenge, *E-model* also provides a lower bound for searching the optimal subset with a greedy algorithm (Section 6.2).

- Based on the *E-model*, we further devise our novel model, *S-model*, which incorporates two important social contexts that influence topic generation and dissemination, namely **CrowdExp** and **Retweet** topic score. The former considers the influence of both the experts and the crowd (the majority of the users) while the latter captures the influence of Twitter users' actions (i.e., retweet in our case) (Section 6.3).

- We conduct experiments on real-world Twitter datasets using two different applications, Topic Discovery and Tweet Classification. The experimental results demonstrate that, by leveraging social contexts, *S-model* can help topic-preserving data summarization and further improve application performance by up to 18% (Section 6.4).

6.2 Data Summarization for Twitter Topics

In this work, we use Twitter data, as an example of social data, to explore topic-preserving data summarization. Thus, in this section, we begin by formally defining our Twitter data model. Then, we discuss statistical topic models in a case of Twitter. Finally, we will introduce our data summarization framework for Twitter topics.

6.2.1 Twitter Data

Here we first give a data model for tweets and then we define topics in the Twitter dataset. Generally, a tweet is represented as a textual feature vector, where each dimension can be constructed using different models, e.g., N-grams, tf-idf scheme, Part of Speech. In this work, given we are targeting to preserve topics, we employ the "bag-of-words" model, following common simplification in most work in information retrieval and topic modeling [139, 140], to construct the feature space using term frequency as the feature weight. Formally, we define a tweet as:

Definition 5. (Tweet): a text tweet t in a Twitter dataset $\mathcal{T}$ is a sequence of words $w_1, w_2, \dots w_{|t|}$, where w_i is a word from a fixed vocabulary $\mathcal{W}$. We represent a tweet with a bag of words, i.e., $t = \{w_1, w_2, \dots, w_{|t|}\}$.

Based on the definition 5, we can define a Twitter dataset (i.e., a collection of tweets) as a Tweet-word matrix:

Definition 6. (Tweet-word matrix): Given a fixed Twitter dataset $\mathcal{T}$ with a collection of N tweets, we can easily build a Tweet-word matrix $R_{|\mathcal{T}|\times|\mathcal{W}|}$, in which each entry $r_{w,t} = n(w,t)$ represents the frequency of word w in tweet t. We use $n(w,t)$ to denote the frequency of word w in t.

A topic in the Twitter dataset is modeled as a distribution over words. Formally,

Definition 7. (Topic): a semantic topic τ in a Twitter dataset $\mathcal{T}$ with a fixed vocabulary $\mathcal{W}$ is represented by a topic model θ, which is a probabilistic distribution of words $\{p(w|\theta)\}_{w\in\mathcal{W}}$. Clearly, we have $\sum_{w\in\mathcal{W}} p(w|\theta) = 1$ and we assume there are altogether K topics in $\mathcal{T}$.

In the following, we will further discuss statistical topic models in detail.

6.2.2 Statistical Topic Models

Compared to tweet-word matrix in Definition 6 that can be derived directly from a Twitter dataset, topics in Definition 7 are latent variables that need to be learned from the dataset. For topic models such as the Probabilistic Latent Semantic Indexing (pLSI) and the Latent Dirichlet Allocation (LDA), the assumptions are that each document is a distribution over topics and each topic is a distribution over words [139, 141]. Specifically, the generative process of a tweet is represented as :

$$p(w,t) = p(t)p(w|t) = p(t)\sum_{\theta\in\Theta} p(w|\theta)p(\theta|t) \tag{6.1}$$

where $p(w,t)$ denotes the probability of observing a word w in a tweet t and can be further interpreted as the production of $p(t)$, the probability distribution of tweets, and $p(w|t)$, the probability distribution of

words given a tweet. For topic modeling, the assumption is that there is a latent topic θ for each word w. Thus, $p(w|t)$ can be further modeled as the multiplication of $p(w|\theta)$, the probability distribution of words given a topic, and $p(\theta|t)$, the probability distribution of topics given a tweet. Given there are a total of $K = |\Theta|$ topics, we sum the multiplication over a set of all independent topics. In this way, we can see that the set of topics Θ is an additional (latent) layer between tweets and words, whose parameters needs to be estimated from the dataset.

Given Eq. 6.1, we can infer $p(w|\theta)$ and $p(\theta|t)$ by maximizing the log-likelihood function of the observed tweet-word matrix

$$L(\mathcal{T}) = \log(\prod_{t\in\mathcal{T}}\prod_{w\in\mathcal{W}} p(w,t)^{n(w,t)}) \tag{6.2}$$

$$= \sum_{t\in\mathcal{T}}\sum_{w\in\mathcal{W}} n(w,t)\log p(w,t) \tag{6.3}$$

$$= \sum_{t\in\mathcal{T}}\sum_{w\in\mathcal{W}} n(w,t)\log(\sum_{\theta\in\Theta} p(w|\theta)p(\theta|t)) \tag{6.4}$$

The goal of a topic model is to estimate the parameters $p(w|\theta)$ and $p(\theta|t)$ which measure the correlation between a word w and a topic θ and that between a topic θ and a tweet t, respectively. For example, if we have $p(w_{football}|\theta_{sports}) > p(w_{football}|\theta_{entertainment})$, it means that the word *football* is more likely to occur in the topic *sports* than in the topic *entertainment*. The parameter space depends on the complexity of different topic models. For pLSI, the parameter space is $O(Kn + Kd)$, where n is the size of Twitter dataset, d is the size of vocabulary and K is the number of latent topics. We can see that the number of parameters shows linear growth in the number of tweets n, which suggests that the model is prone to overfitting [139]. To overcome the overfitting problem, LDA treats the topic weights as a K-parameter hidden random variable (i.e., follows Dirilecht distribution) rather than a large set of individual parameters that are linked to each dataset. In this way, the parameter space of LDA model is $O(K + Kd)$ which does not increase linearly in the size of dataset. Therefore, LDA does not suffer from the overfitting problem like pLSI.

6.2.3 Topic-preserving Data Summarization

Given a Twitter dataset $\mathcal{T}$, we aim to select a subset $\mathcal{S} \subseteq \mathcal{T}$ of bounded size $|\mathcal{S}| = k$, which maximizes the objective function $F : 2^{|\mathcal{T}|} \to \mathbb{R}$.

$$\mathcal{S}^* = \arg\max_{\mathcal{S}\subseteq\mathcal{T},|\mathcal{S}|=k} F(\mathcal{S}) \tag{6.5}$$

where we define S^* as the *summarized subset* of the original dataset $\mathcal{T}$. If we know the utility function F, the problem as shown in Eq. 6.5 is the classical knapsack problem which can be solved via greedy approximation algorithm illustrated in Algorithm 6.1. In our case, our target is to select a subset that preserves the topics in the original dataset. Thus, based on Eq. 6.4 of topic modeling, we formulate the objective function of topic-preserving data summarization as:

$$F(\mathcal{S}) = L(\mathcal{S}) = \sum_{t\in\mathcal{S}}\sum_{w\in\mathcal{W}} n(w,t)\log(\sum_{\theta\in\Theta} p(w|\theta)p(\theta|t)) \tag{6.6}$$

where the parameters $p(w|\theta)$ and $p(\theta|t)$ need to be estimated based on the Twitter dataset as discussed before. Thus, from Eq. 6.6, we discover that there are mainly two challenges to preserve latent topics

in the Twitter dataset: **C1-parameter challenge**: topic-related parameters (i.e., $p(w|\theta)$, $p(\theta|t)$) in the objective function are unknown, which need to be estimated based on each subset; **C2-combination challenge**: the search space of the number of possible subsets is exponential and even with the cardinality constraint of size k, we would still need to search over $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ to determine $\mathcal{S}^*$.

The first challenge **C1** is caused by the complexity of topic models. Recall that for Eq. 6.3, we can see that the objective is to maximize total log-likelihood log$p(w,t)$ of picking the cell $n(w,t)$ from the tweet-word matrix. Eq. 6.4 incorporates the topic modeling of pLSI in Eq. 1. As is discussed earlier, the parameter space of pLSI is $O(Kn + Kd)$ while that of LDA is $O(K + Kd)$. The immediate idea to overcome the parameter challenge is that we can employ a simple topic model, which has fewer parameters. In [139], LDA and pLSI are compared with other simpler models, i.e., unigram and mixture of unigrams. Inspired by this, we first consider exploiting the unigram model for data summarization to preserve the topics in Twitter datasets. The unigram model assumes that the words of every tweet are drawn independently from a single multinomial distribution:

$$p(w,t) = \prod_{w \in t} p(w) \tag{6.7}$$

where $p(w) = \frac{\sum_{t \in \mathcal{T}} n(w,t)}{\sum_{t \in \mathcal{T}} \sum_{w \in \mathcal{W}} n(w,t)}$. Here, we can see that the parameter space of the unigram model is $O(d)$ where d is the size of vocabulary. By incorporating the unigram model of Eq. 6.7 into Eq. 6.3, we have unigram-weighted summarization model

$$F(\mathcal{S}) = \sum_{t \in \mathcal{S}} \sum_{w \in \mathcal{W}} n(w,t) \text{log} p(w) \tag{6.8}$$

Compared to pLSI formulation in Eq. 6.6, Eq. 6.8 reduces the parameter space from $O(Kn + Kd)$ to $O(d)$ i.e., only bounds to the size of vocabulary. However, we notice that unigram-weighted summarization model prefers to select the tweets with high-frequency words. This leads to summarization bias, as only those topics which correspond to high frequency words will be preserved. Motived by information theory, we argue that topics are evenly distributed across tweets (i.e., high entropy) rather than concentrated in a few tweets (i.e., low entropy). Thus, we design an objective function based on information entropy to evaluate the information that is contained in each selected tweet and we formulate this data summarization model as:

$$F(\mathcal{S}) = \sum_{t \in \mathcal{S}} \sum_{w \in \mathcal{W}} n(w,t) p(w) \text{log} \frac{1}{p(w)} \tag{E-model}$$

With this formulation, *E-model* is more likely to select a subset with high entropy. It can be interpreted as the higher the entropy of a subset, the more information or topics it contains. Meantime, entropy is also a measure of *uncertainty* or *diversity*, which can reduce summarization bias compared to unigram-weighted model in Eq. 6.8. We can solve *E-model* by greedy algorithm as shown in Algorithm 6.1. In addition, the *E-model* presents three nice properties that help us solve the combination challenge **C2**.

The first property of *E-model* is monotonicity. That is, addition of more tweets to an existing subset will increase the utility of the overall selection.

Proposition 1. (Monotonicity). Let $\mathcal{T}$ be a collection of tweets, $\mathcal{S}^* =< t_1, \cdots, t_n >, t_i \in \mathcal{T}, 1 < i < n$ a selection, and $t' \in (\mathcal{T} \backslash \mathcal{S}^*)$ is from a set of non-selected tweets. Then it holds that: $F(\mathcal{S}^* \cup \{t'\}) \geq F(\mathcal{S}^*)$

Algorithm 6.1: Greedy algorithm for E-model

Require: $\mathcal{T}$: original dataset, k: cardinality constraint
Ensure: $\mathcal{S}$: summarized subset
1: $\mathcal{S} \leftarrow \emptyset$
2: **while** $|\mathcal{S}| \leq k$ **do**
3: $L = \{t \in \mathcal{T} \setminus \mathcal{S}\}$
4: $t = \arg\max_{t \in L} F(\mathcal{S} \cup \{t\}) - F(\mathcal{S})$
5: $\mathcal{S} = \mathcal{S} \cup \{t\}$
6: **end while**
7: **return** $\mathcal{S}$

Proof. We denote all the words that occurs in t' but not in current selection $\mathcal{S}^*$ as the set of words $\{t' \backslash \mathcal{S}^*\}$. Then, we have $F(\mathcal{S}^* \cup \{t'\}) - F(\mathcal{S}^*) = \sum_{w \in \{t' \backslash \mathcal{S}^*\}} n(w, t')p(w)\log\frac{1}{p(w)} \geqslant 0$. If all the words in the t' have already occurred in the current selection T^*, i.e., $\{t' \setminus \mathcal{S}^*\} = \emptyset$, the utility will not increase. □

Second, *E-model* shows submodularity which refers to the property that marginal gains start to diminish due to saturation of objective. That is, adding a tweet to a smaller set helps more than adding it to a larger set, w.r.t. the size of summarized subset.

Proposition 2. (Submodularity). Let $\mathcal{T}$ be a collection of tweets, $\mathcal{S}^* =< t_1, \cdots, t_n > t_i \in \mathcal{T}, 1 < i < n$ a selection, and $t, t' \in (\mathcal{T} \setminus \mathcal{S}^*)$ a set of non-selected tweets. Then it holds that: $F(\mathcal{S}^* \cup \{t\}) - F(\mathcal{S}^*) \geq F(\mathcal{S}^* \cup \{t'\} \cup \{t\}) - F(\mathcal{S}^* \cup \{t'\})$.

Proof. since we have $\{t \setminus \mathcal{S}^*\} = \{t \setminus (t \cap \mathcal{S}^*)\} \supseteq \{t \setminus (t \cap (\mathcal{S}^* \cup t'))\} = \{t \setminus (\mathcal{S}^* \cup \{t'\})\}$, then we can derive
$F(\mathcal{S}^* \cup \{t\}) - F(\mathcal{S}^*) = \sum_{w \in \{t \backslash \mathcal{S}^*\}} n(w, t)p(w)\log\frac{1}{p(w)} \geqslant$
$\sum_{w \in \{t \backslash (\mathcal{S}^* \cup \{t'\})\}} n(w, t)p(w)\log\frac{1}{p(w)} = F(\mathcal{S}^* \cup \{t'\} \cup \{t\}) - F(\mathcal{S}^* \cup \{t'\})$. Here we can see that the equality holds if and only if $\{t \cap t'\} = \emptyset$. □

Thus, based on Proposition 1 and 2, Algorithm 6.1 provides the performance guarantee for searching the optimal subset with *E-model*.

Proposition 3. (Near-Optimality). Algorithm 6.1 is a (1-1/e)-approximation to the optimal value for $E - model$.

Proof. For any monotone, submodular function F with $F(\emptyset) = 0$, it is known that a greedy algorithm that selecting the element t with the maximal value of $F(\mathcal{S} \cup \{t\}) - F(\mathcal{S}^*)$ with T as the elements selected so far has a performance guarantee of $(1 - 1/e) \approx 0.63$ [142]. This result is applicable to Algorithm 6.1, since the objective function of *E-model* is monotonic (Proposition 1) and submodular (Proposition 2) with $F(\emptyset) = 0$. □

In the next section, we will further discuss that how we use these properties to solve the combination challenge **C2**.

6.3 Optimization

Under the previous *E-model*, we design objective functions with only considering the correlation between words and topics. However, Twitter data not only has rich text information but also the contextual information about users (e.g., number of followers, location) and their actions (e.g., reply, retweet, favorite, block). Summarizing Twitter data with *E-model* ***neglects*** the correlation between topics and the users who generate and disseminate topics. In the following, we will firstly enhance *E-model* with leveraging social contexts. Then we improve Algorithm 6.1 via lazy evaluation to improve search efficiency over an exponentially space.

6.3.1 An Enhanced Model with Social Contexts

Our task is to summarize Twitter dataset with a small subset while preserving the topics in the original dataset. The task is challenging since a topic is a latent concept. In the previous *E-model*, we measure the correlation between words and topics with unigram weights and entropy. However, we did not consider that a social topic is generated from tweets that are written and retweeted by users in a social network. Thus, the generative process of a topic is highly correlated to users and their actions such as retweet, like and replyto. In the following, we firstly discuss two straightforward social contexts in the Twitter, namely: user influence and tweet influence. Then based on understanding of these two simple social contexts, we present our design of two topic scores.

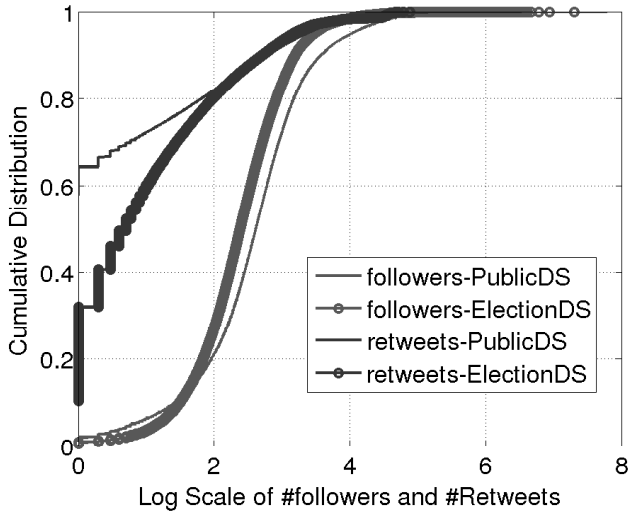

Figure 6.1: Cumulative distribution of the number of followers and the number of retweets in log scale (base 10)

User influence can be measured in various ways including PageRank, Granger Causality, Node's Indegree, etc [143] while tweet influence can be measured by the number of retweets or replies, etc. In this work, we model user influence in terms of the number of followers while tweet influence in terms of the number of retweets. To better understand user and tweet influence, we plot the cumulative distributions of both the number of retweets of a tweet message and the number of followers of a user based on two Twitter datasets (detailed in Section 6.4.1). It clearly shows that only about 10% of users

have more than 3000 followers (with the highest having about 3.5 million followers) while the majority of users (about 80%) have less than 1200 followers. As for the number of retweets, we observe that only about 5% of tweets are retweeted more than 3000 times while the majority of tweets (about 60%) are retweeted less than 10 times. From Figure 6.1, it can be also seen that there are two extremes in the number of followers, while the majority of the users lies in the middle (70% of them have followers in the range [50,1200]). Thus, we can deduce that *topics are largely generated by a crowd of users who have about 50-1200 followers and their tweets are normally retweeted less than 10 times*. In other words, we should assign more weights to the crowd (the majority of the users) for their effects on the topic generation and dissemination, not just giving more weights to those users with a high number of followers and those tweets with a high number of retweets.

Base on these observations from Figure 6.1, we further design two topic scores to capture the impact of the crowd, the experts and their retweet actions on topic generation and dissemination, namely:

(1) CrowdExp topic score: We use CrowdExp topic score to measure the contribution of a user on generating the latent topics. Intuitively, the higher the influence of a user, the more contribution she makes. In other words, we would like to preserve those tweets that are posted by topic experts who have the maximum number of followers. Meanwhile, according to [144], a group of diverse crowd can outperform a group of experts due to the effect of collective wisdom. Also, based on our discussion before, we know that most of the topics in Twitter are generated by the majority of the users. Thus, we seek to devise a topic score that considers the influence of both topic experts and the majority of the users, on topic generation and dissemination. We name our topic score, *CrowdExp*. As the name suggests, CrowdExp, is a combination of the influence that the crowd of users has on topics in a dataset, and the experts, those that have a high number of followers. The influence of users in social network can be evaluated by various algorithms such as PageRank and Granger Causality. However, these algorithms are too expensive to compute. In this work, we simply model user influence as the number of followers x, i.e., in-degree of a node in a user relation graph. There are two steps to compute the CrowdExp topic score: distribution estimation of user influence and topic score computation.

Step 1: Distribution estimation of user influence. Figure 6.1 shows that the number of followers of Twitter users x follows a power-law distribution, the probability density function of which is defined by

$$f_u(x) = (\alpha - 1)(1 + x)^{-\alpha} \tag{6.9}$$

where α is the exponent parameter and can be derived by maximum likelihood estimators (MLEs)

$$\alpha = 1 + |X|[(\sum_{x \in X} ln(1 + x))]^{-1} \tag{6.10}$$

in which X is the set of observed values, i.e., the number of followers for users that write tweets. Furthermore, we compute user influence based on the cumulative distribution function over $f_u(x)$

$$F_u(x) = \int_0^x f_u(z)dz = 1 - (1 + x)^{(1-\alpha)} \tag{6.11}$$

In this way, the user influence is mapping to the interval $[0, 1]$ in which a topic expert has the value of influence approaching to 1.

Step 2: CrowdExp topic score computation. In this step, we aim to design a score function that models the influence of both the majority crowd and the experts. We apply the following piecewise function which takes user influence as the input to compute CrowdExp topic score:

$$u_t = \begin{cases} -F_u(x)\log F_u(x), x \leq F_u^{-1}(\eta) \\ \log(F_u(x) + \phi), x > F_u^{-1}(\eta) \end{cases} \tag{6.12}$$

where η is the cut-point of experts and non-experts and ϕ is a location parameter. For example, if $\eta = 0.9$, it means that the experts are those top 10% of users who have a high number of followers. In practice, we tune this parameter from 0.85 to 0.95 in order to find the suitable value that has the best performance in terms of application metrics. With this function, we can give more weights for the crowd who are majority and the experts who are important.

(2) Retweet topic score: Similar to user influence, we can also evaluate the importance of a tweet by the number of retweets. 'Retweet' happens when a user is interested in some topics that are contained in a tweet and thus decides to further disseminate it. In other words, retweet reflects approval or recommendation that a tweet has in a community, which is an important social context [145]. We infer that there are two correlations that between topics and retweet actions: 1) most topics are generated from the majority of tweets (from Figure 6.1 it can be seen that the majority of tweets is retweeted that less than 10 times); 2) the higher the number of retweets, the more likely a tweet can become a topic. We apply similar process to compute the tweet importance based on the number of retweets y. Firstly, we estimate the scale parameter β for probability density function:

$$f_r(y) = (\beta - 1)(1 + y)^{-\beta} \tag{6.13}$$

Then, we compute tweet importance based on the cumulative distribution function over $f_r(y)$

$$F_r(y) = \int_0^y f_r(z)dz = 1 - (1 + y)^{(1-\beta)} \tag{6.14}$$

Finally, we design retweet score function as an absolute log function to capture two correlations that discussed above.

$$re_t = |\log(F_r(y) + \gamma)| \tag{6.15}$$

where γ is a location parameter and is determined by cut-point of majority and non-majority. From this function, we can observe that more weights are given to the tweets which are important and those which are the majority.

(3)Put All Together. Finally, we incorporate the CrowdExp and retweet topic scores, given by Equations 6.12 and 6.15, into *E-model* based on the product rule. Then, we have an enhanced model with exploiting social contexts *S-model* that

$$F(\mathcal{S}) = \sum_{t \in \mathcal{S}} \sum_{w \in \mathcal{W}} n(w,t) \cdot \left[u_t \cdot re_t \cdot p(w) \cdot \log \frac{1}{p(w)} \right] \tag{S-model}$$

Thus, we can see that *S-model* considers the impact of both experts and majority users, and their retweet actions on topic generation and dissemination, as well as the content diversity based on entropy measures for topic-preserving data summarization.

Also, it is easy to discover that *S-model* preserves all the properties of *E-model* and thus we can also solve *S-model* by Algorithm 6.1. However, for a large dataset with the huge number of items, the time complexity of Algorithm 6.1 is high. Next, we will discuss how we optimize Algorithm 6.1 by applying lazy evaluation.

Algorithm 6.2: Lazy greedy algorithm

Require: $\mathcal{T}$: original dataset, k: cardinality constraint
Ensure: $\mathcal{S}$: summarized subset
1: $\mathcal{S} \leftarrow \emptyset$
2: $PriorityQueue < Node > queue$ //store marginal gains
3: **for** $t \in \mathcal{T}$ **do**
4: $marginalGain = F(\mathcal{S} \cup \{t\}) - F(\mathcal{S})$
5: queue.add(t, marginalGain) //initial $\rho(t)$
6: **end for**
7: **while** $|\mathcal{S}| \leq k$ **do**
8: t = queue.poll() // get the top tweet
9: $marginalGainCurrent = F(\mathcal{S} \cup \{t\}) - F(\mathcal{S})$
10: **if** marginalGainCurrent ¿ $\rho(t)$ **then**
11: $\mathcal{S} = \mathcal{S} \cup \{t\}$
12: **else**
13: queue.add(t, marginalGainCurrent)
14: **end if**
15: **end while**
16: **return** $\mathcal{S}$

6.3.2 Lazy Greedy

Given a tweet dataset $\mathcal{T}$, Algorithm 6.1 needs $O(|\mathcal{T}| * k)$ times evaluation on function $F(\cdot)$ in order to find an optimal summarized subset of size k. However, when the size of $\mathcal{T}$ increases greatly, the standard greedy algorithm becomes more expensive. Fortunately, Minoux [146] *et.al* developed lazy greedy algorithm, which exploits submodularity to prune the search space and accelerate the searching. Since our objective function is submodular, we can apply this technique and Algorithm 6.2 shows our implementation of lazy greedy. Instead of computing $F(\mathcal{S}^* \cup \{t\}) - F(\mathcal{S}^*)$ for each tweet $t \in \mathcal{T}$, the lazy greedy algorithm keeps an upper bound $\rho(t)$ on the marginal gain sorted in decreasing order (i.e., store in a priority queue in Algorithm 6.2). In each iteration, the lazy greedy evaluates the element on top of the list (line 8-9), say t, and updates its upper bound (line 10-14). If after the update $\rho(t) \geq \rho(t')$ for all $t' \neq t$, submodularity guarantees that t is the element with the largest marginal gain. Even though the exact cost (i.e., number of function evaluations) of lazy greedy is unknown, this algorithm leads to orders of magnitude speedups in practice [147].

6.4 Experiments

In this section, we empirically evaluate the *S-model* in terms of both computational cost and performance using two real-world Twitter datasets. Through extensive experiments, we aim to answer the following questions:

- How effective is the proposed *S-model*, compared to other existing solutions which do not consider social contexts, in preserving topics in the Twitter dataset?
- How is the performance of different applications based on topic models influenced by different summarization models?
- What are the tradeoffs between cost and performance entailed in data summarization?

In the following, we first introduce the datasets and then present our experimental settings and results.

6.4.1 Datasets

We use two datasets crawled from Twitter based on two different public APIs: Sample streaming API and Search REST API [148]. The former focuses on completeness while the later on relevance [149]. The first dataset, which we denote as **PublicDS**, is collected using the public Sample streaming API which returns a small random sample (about 1%) of all public real-time tweets. We select 1 million Tweets which are written in English. The second dataset, which we denote as **ElectionDS**, is about the US 2012 Presidential Election, collected using the public Search API. The Search API allows users to specify one or more search terms to obtain historical tweets and only those tweets that match the specified terms are returned. For example, for the 2012 presidential election, we can use political keywords such as "Barack Obama", "Mitt Romney" or "Joe Biden", etc. ElectionDS also contains 1 million tweets. Table 6.1 summarizes the statistics of two datasets.

	PublicDS	**ElectionDS**
#Users	716,958	634,814
#Tweets	1 million	1 million
#Max in-degree	61,803,119	20,143,264
#Max retweets	3,330,699	76,096

Table 6.1: Statistics of two Twitter datasets

6.4.2 Experimental Setup

Evaluated techniques: We compare *S-model* with existing solutions that do not exploit social contexts:

- *E-model*: It considers the information entropy as also shown in Section 6.2.3.
- *Random*: It is the most commonly used sampling methodology that selects items randomly from original big datasets, as it preserves certain important statistical properties of the entire dataset. It is also used in Twitter's public stream API which returns about 1% random sample of all tweets.
- *ClusterSum*: In this technique, the objective function of *K-medoids* clustering is used, which defines representativeness of an item (i.e., how well a selected item represents the other items in the dataset) based on distance measures. For example, the loss function $L(S) = \frac{1}{|V|}\sum_{v \in V} \min_{s \in S} d(v, s)$ is to find a subset $S \subset V$ of medoids that has a minimum average distance (e.g., cosine distance in our evaluation) to each element in the original dataset V. For details, please refer to [150].
- *KLSum*: KLSum generates a summary S given a tweet collection D via the objective function $S^* = \min_{s \in D} KL(P_D||P_S)$, where KL is Kullback-Lieber (KL) divergence [151]. This method greedily selects tweets to a summary set so long as it decreases the KL divergence.
- *SvdSum*: This method represents tweet collection D as a matrix and applies singular value decomposition (i.e., $D = U\Sigma V^T$) to text summarization [152]. This method selects the most informative tweets according to the matrix V^T.
- *LexSum*: LexSum assesses the eigenvector centrality of each tweet and extracts the most important ones to include in the summary [153].

Metrics: we evaluate data summarization models in terms of both computational cost and performance. From the cost perspective, we focus on time cost and break it down into two phases: 1) *Summarization Time* to complete data summarization and 2) *Training Time* to finish training topic models based on summarized subsets. Both summarization and training time costs depend on the size of summarized subsets. Besides the time cost, we still need to consider the performance of different data summarization models, where performance is defined by different applications. A good data summarization model shall not only spend less time on summarizing the dataset but also provide good performance for applications based on summarized subsets.

From the performance perspective, we analyze two representative applications based on topic models. The first application is Topic Discovery. In this application, the baseline performance is represented by the topics generated from the original dataset. The performance of different summarized subsets (based on different summarization techniques) is evaluated by comparing their divergence from the baseline. Since a topic is a probability distribution over words, we use Jensen-Shannon divergence (JSD) to measure the distance between two topics i.e, divergence between two probability distributions. Specially, we compute JSD as

$$JSD(\tau_1||\tau_2) = \frac{1}{2}[KL(\tau_1||M) + KL(\tau_2||M)] \tag{6.16}$$

where $M = \frac{1}{2}(\tau_1 + \tau_2)$, τ_1 and τ_2 are two topics, and KL represents the Kullback-Leibler divergence. A low value of JSD means that two topics (probability distributions) are highly similar, i.e., topics are well preserved.

The second application is Tweet Classification. The goal of this application is to classify tweets with the same topics together. We use hashtags and search terms as the ground truth for classification and further compute the classification accuracy. The higher the classification accuracy, the better the summarized subset.

6.4.3 Evaluation

In the following, we will detail our evaluation from the perspective of both computational cost and performance.

6.4.3.1 Computational Cost

The time cost largely depends on the size of summarized subsets. We define *summarization ratio* as the size of summarized subset $\mathcal{S}$ to that of the (entire) original dataset $\mathcal{T}$, i.e. *summarization ratio*$= |\mathcal{S}|/|\mathcal{T}|$. We vary summarization ratio from 0.005 to 0.04 with *S-model* and Figure 6.2 shows the results based on the PublicDS. We can see that with the increase in the size of summarized subset, the summarization time increases slowly. Let's consider the case in Figure 6.2 (a) where the summarization ratio varies from 0.01 to 0.04. We can see that the summarization time only increases by 1 second, from about 9.1 seconds to 10.1 seconds while the size of summarized subset increases three times from about 1 percent to 4 percent of the original dataset. On the other hand, we observe the training time cost increases linearly in the size of summarized subsets from 24 seconds at 0.01 to about 97 seconds at 0.04. If we train the topic models based on the entire original dataset, the training time is about 3820 seconds (not shown in Figure 6.2). In addition, Figure 6.2 (b) also shows that if we do not apply lazy evaluation, the time cost of data summarization is highly expensive. As we can observe, it takes almost 3000 seconds to

select 4000 tweets from a total of 1 million tweets (i.e., summarization ratio=0.004). Thus, our results confirm that lazy evaluation brings about orders of magnitude improvement in the summarization time cost. Thus, based on our observations, we discover that sum of the time cost on data summarization and training times on the summarized subsets of our proposed *S-model* is negligible compared to the high training cost without data summarization. Also, we notice that the time cost on training models based on summarized subsets increases proportionally to the summarization ratio.

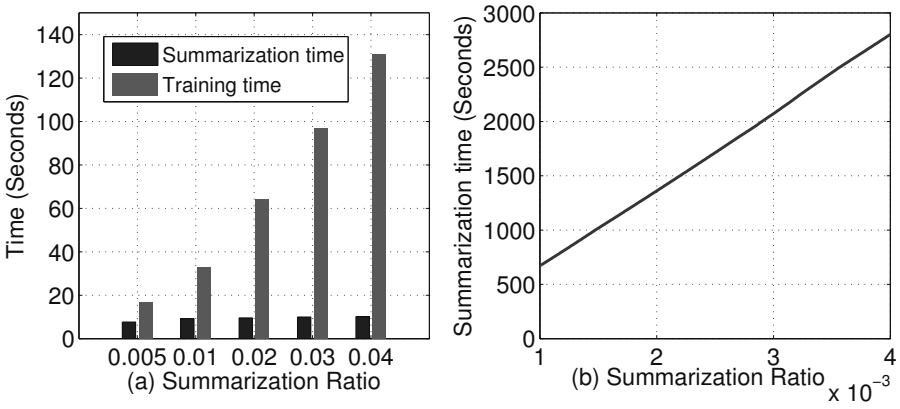

Figure 6.2: (a) Time cost on data summarization with S-model and topic model training with LDA; (b) Time cost without lazy evaluation of S-model

Meanwhile, we note that the time costs of *E-model* is highly similar to that of *S-model*. The overhead of *S-model* to compute CrowdExp and Reteet topic score is negligible (less than 1 second on both datasets). In addition, the results of time cost of *S-model* based on ElectionDS show a similar trend, which we omit for saving space. Finally, we discover that the summarization time cost of all other baselines (except Random) are an order of magnitude more computationally intensive, due to lack of optimizations for summarizing large-scale datasets.

6.4.3.2 Performance

In the above experiments, we demonstrated that data summarization can reduce the time cost on training models. Next, we will evaluate the performance of different summarization models for two applications, namely Topic Discovery and Tweet Classification. For performance evaluation of Topic Discovery, we compute the difference between topics generated on the basis of summarized subsets, and those that are generated based on the original dataset. For Tweet classification, we evaluate how different data summarization methods influence application's performance in terms of classification accuracy.

Topic Discovery: LDA is a widely used topic model to discover the topics in the Twitter dataset [154, 155, 156]. It takes three parameters as its input: the number of topics K, a hyperparameter for the Dirichlet prior topic distribution α and a hyperparameter for the Dirichlet prior word distribution β. Choosing optimal parameters is a very challenging problem, and it is not the focus of our work. In this work, we set $K = 100$ and use priors of $\alpha = 50/K$, and $\beta = 0.01$ as suggested by [139, 157]. We use Mallet software library [158] to discover the topics in the dataset. We firstly discover the topics based on the entire original dataset and we denote this set of topics as $\Gamma^{full} = \{\tau_1^{full}, \cdots, \tau_{100}^{full}\}$. Then

based on different summarization models, we discover the topics respectively and we denote this set as $\Gamma^{sum} = \{\tau_1^{sum}, \cdots, \tau_{100}^{sum}\}$.

Since there is no implicit orderings of LDA's topics, we first match them based on the similarity of the words in the distribution. To match the topics, we construct a weighted bipartite graph between the topics from Γ^{full} and Γ^{sum}. Since each topic is a bag of words, we can apply the Jaccard similarity between the words in two topics from different topic sets as the weight of the edges in the graph.

$$d(\tau_i, \tau_j) = \frac{|\tau_i \cap \tau_j|}{|\tau_i \cup \tau_j|}, \tau_i \in \Gamma^{full}, \tau_j \in \Gamma^{sum}$$

After constructing the graph, we use maximum weight matching algorithm proposed in [159] to find the best matches between topics from Γ^{full} and Γ^{sum}. After matching, we can compute Jensen-Shannon divergence using Eq. 6.16 between topics since each topic is a probability distribution over words.

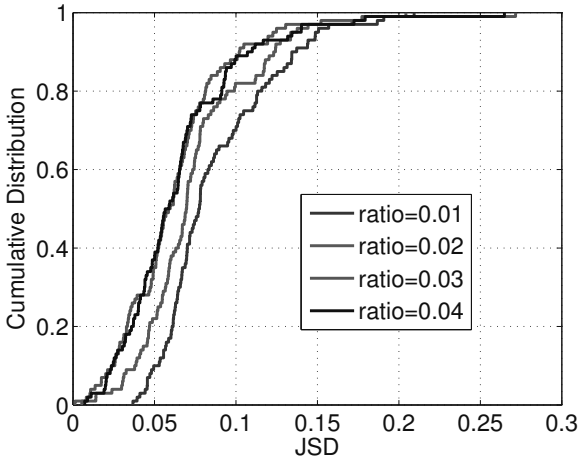

Figure 6.3: JSD with varying summarization ratio of S-model

Before we evaluate the performance of different models, we firstly evaluate the effect of summarization ratio by varying its value from 0.01 to 0.04 only with *S-model*. Then, we compute the JSD between topics that are generated based on the original dataset and those generated from summarized subsets of different summarization ratios. Figure 6.3 shows the results. It is clear that with more data in the summarized subsets, the topic distribution generated by summarization is closer to that generated based on the original dataset. However, we can also observe that more data provides diminishing benefits since the summarized subsets with $ratio = 0.03$ and $ratio = 0.04$ show very similar results.

For the following set of experiments, we fix the summarization ratio to 0.02 to compare the performance of different models. For each model, we compared the topics generated from the subset (with ratio 0.02) to the topics generated from the original dataset. Figure 6.4 and Figure 6.5 show the histograms of JSD of different models based on PublicDS and ElectionDS, respectively. We observe clearly that both *E-model* and *S-model* outperform other models. In detail, we can see that *E-model* has the lowest average JSD (i.e., $\mu = 0.0666$ for PublicDS and $\mu = 0.0684$ for ElectionDS) while *S-model* has the lowest standard deviation of JSD (i.e., $\sigma = 0.035$ second lowest for PublicDS and $\sigma = 0.0384$ for ElectionDS). In addition, we observe that only all JSD of *S-model* in Figure 6.4 (a) and Figure 6.5 (a) are less than 0.2 while there always exist some topics (i.e., with $JSD > 0.2$) that are far from the topics

generated based on Γ_{all} for all other models. We infer the reason is that compared to other models like *E-model* which only consider the correlation between words and topics, the performance of *S-model* is further regularized by social contexts that are exploited in *S-model*. That is also why other models have higher probability to choose some outliers, which can be observed in Figure 6.4 (b)-(g) and 6.5 (b)-(g) with some JSDs larger than 0.2. Another interesting observation is that the performance of *LexSum* and *ClusterSum* are even worse than Random model. We conjecture this is because the topics are evenly distributed in the dataset rather than being clustered around some representative tweets (in terms of similarity or centrality measures). In such a scenario, it can be expected that Random model would do better than *LexSum* and *ClusterSum*.

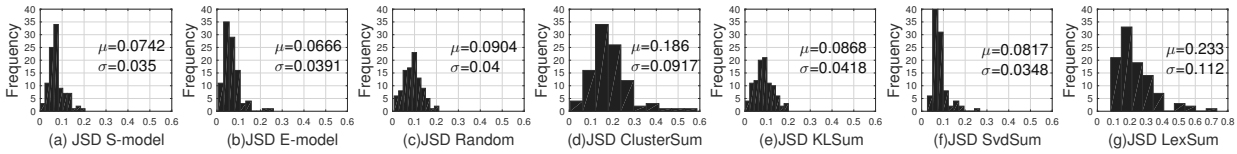

Figure 6.4: Histograms for JSD based on PublicDS

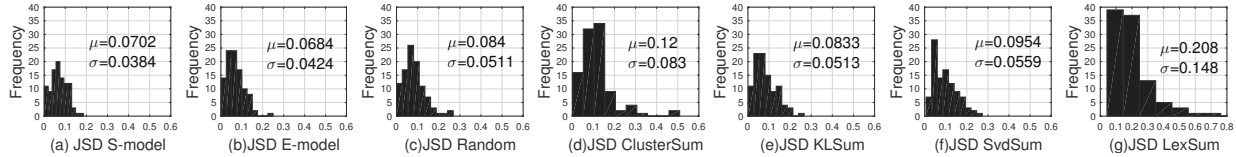

Figure 6.5: Histograms for JSD based on ElectionDS

From above results, we can safely reach two conclusions: 1) *S-model* shows very similar performance to the *E-model* with the added benefit that it is less likely to select the outliers from the original dataset; 2) Representativeness-based approaches which are based on similarity (used in ClusterSum) or centrality (used in LexSum) measures show poor performance in preserving topics in the dataset.

Tweet Classification: Based on an already trained topic model, we can estimate the topic probability for a new tweet and further classify tweets within the same topic together. For PublicDS, we select a set of 100 hashtags, each of which is a label for a unique category. Further, we select 500 new tweets for each hashtag (i.e., tweets labeled with particular hashtags). Then, given a new tweet, we infer its topic distribution and assign it to a topic that has a maximum probability distribution. We thus have tweets which have originally been labeled by particular hashtags, and subsequently, have topics assigned to them. Finally, for each hashtag, we count the number of tweets belonging to that hashtag, which have been assigned to the same topic. Ideally, if all the tweets with the same hashtag are assigned to the same topic, the accuracy is 100%. In practice, for tweets belonging to each hashtag, we consider the top 5 topics with the highest probability distribution (as opposed to only one maximum). Then we count the number of tweets that have been assigned to any of these top 5 topics. After this we can define classification accuracy as the number of tweets in the top 5 topics to the total number of tweets (with the same hashtag). As for ElectionDS, instead of using hashtags, we use search terms as category labels. For example, we regard the set of tweets that are retrieved with the same search term as in the same category.

We firstly evaluate the performance of different models by fixing summarization ratio as 0.02 and Table 6.2 shows the accuracy of tweet classification. It is clear that the *S-model* outperforms other data summarization methods for both datasets. In addition, we compare the results of different models

	PublicDS		**ElectionDS**	
	accuracy	p-value	accuracy	p-value
AllData	0.6660	–	0.4648	–
S-model	**0.7133**	<0.0001	**0.5295**	<0.0001
E-model	0.6605	0.5685	0.4702	0.0724
Random	0.6545	0.2166	0.4496	0.0730
ClusterSum	0.6514	0.0193	0.4311	<0.0001
KLSum	0.6553	0.7758	0.4533	0.6437
SvdSum	0.6546	0.6841	0.4532	0.6017
LexSum	0.6511	0.7519	0.4362	0.3138

Table 6.2: Classification accuracy that are based on different summarization models and significance analysis with p-values of t-test (α = 0.05)

with the results based on the entire original dataset (i.e., *AllData* in the Table 6.2). We compute p-value using Student's t-test for statistical significance analysis (with 5% significance level). From the Table 6.2, we can observe that (i) both results of *S-model* are significantly better than the results from the entire original dataset for both datasets (with both p-values less than 0.05); (ii) both the results of ClusterSum are significantly worse than that of AllData (with p-values smaller than 0.05) while those of other baselines are not significantly different from the results of AllData (with p-values larger than 0.05). Based on above observations, we can safely conclude that our model that exploits social contexts can effectively preserve topics in the original dataset and also outperforms the other existing summarization methods (those using similarity or centrality measures) for our target applications.

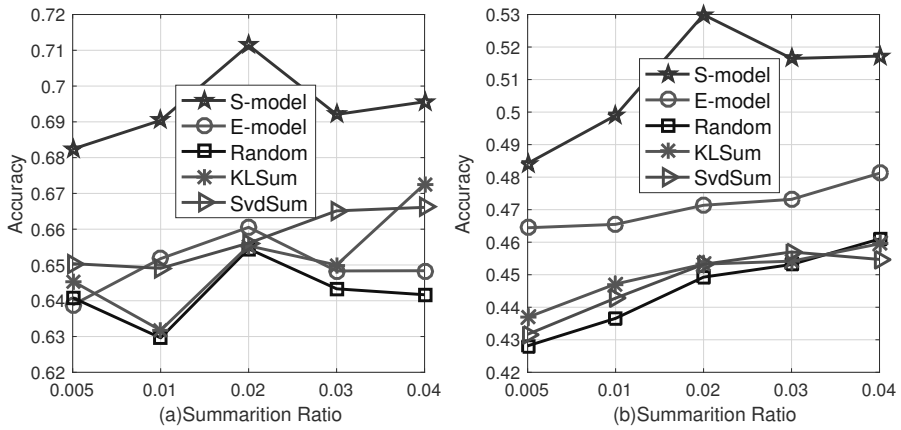

Figure 6.6: Classification accuracy of different models with varying summarization ratios for (a) PublicDS and (b)ElectionDS

Furthermore, we vary the summarization ratio from 0.005 to 0.04 and Figure 6.6 shows the results (we omit *ClusterSum* and *LexSum* since their performance are worse than Random model). It is clear that the performance of *S-model* always outperforms other models at any summarization ratio. With the summarization ratio increasing from 0.005 to 0.02, the overall performance of each model increases slightly for both datasets. However, we still observe that the performance of different models fluctuate as the summarization ratio increases from 0.02 to 0.04, which confirms that more data does not always

improve the application's performance.

6.4.4 Summary

Based on extensive experiments, we can answer the three questions raised in the beginning of this section. From the first application of Topic Discovery, we confirm that our proposed *S-model* which leverages social contexts is effective in preserving topics in the Twitter dataset and is less likely to select outliers than other models which do not consider social contexts. For the second application of Tweet Classification, we observe that the *S-model* outperforms other models. In particular, we observe that the performance of Tweet Classification based on summarized subsets via *S-model* is significantly better than that based on the entire dataset. It means that our *S-model* can effectively select the data that is truly useful for the application as well as being effective in removing noise.

There are two reasons why our proposed *S-model* outperforms other existing methods for preserving topics. First, our model exploits social contexts, i.e., CrowdExp and Retweet topic scores, which model the impact of both centrality and majority in users and their retweet actions, on topic generation and dissemination. This is different from the existing methods which separately utilize the user centrality (LexSum), or similarity-based representativeness (ClusterSum) for data summarizations. Second, *S-model* is enhanced based on *E-model*, which is more likely to generate a subset of highly diverse content. In other words, our proposed *S-model* is novel since it integrates both experts and majority users, their retweet actions, and content diversity based on entropy measures, for social data summarization.

However, as for making sound tradeoff decisions between cost and performance, it largely depends on the performance requirements of different applications. For instance, if the application designer considers 66% accuracy for PublicDS to be good enough for Tweet Classification, then *Random* would be the best solution since the cost of random sampling is almost zero. For our *S-model*, the cost of data summarization is about 10 seconds as shown in Figure 6.2 but the performance of Tweet Classification can be improved by about 9% for PublicDS and 18% for ElectionDS compared to Random method. Thus, our proposed *S-model* will be preferred by applications that have high performance requirements.

6.5 Related Work

Topic model. Topic model [139] is a popular statistical model for discovering topics in text corpora. Previous work [154, 155, 156] discovered the topics with considering both geographic location and time. In addition, based on trained topic models, many applications were developed such as tweet classification [160] and collaborative filtering [139]. In this chapter, we do not focus on new topic models and applications. Instead, we focus on topic-preserving data summarization and we further validate the effectiveness of our approach using two applications that are based on topic models.

Text and Data Summarization. There is a lot of related work on text summarization, which focuses on constructing summaries for a natural language text such as documents [73] and news articles [70]. Unlike text summarization, data summarization aims to identify a truly useful subset of structured data from the original dataset. The 'usefulness' is defined based on data properties such as representativeness [69, 70], diversity [71], informativeness [72] and coverage [73]. However, these work do not exploit social contexts. As for text summarization, many related work focus on different dimensions such as user's sentiments [136], events[137, 138] and contextual information [143]. Specifically, Chang *et.al*

[143] proposed a supervised learning framework to summarize contextual information using different user influence models. Unlike their work on text summarization to generate contextual information, we focus on data summarization to generate a subset of the original dataset while preserving topics. In addition, we consider cost-performance tradeoffs in data summarization, which avoids using compute-intensive learning models.

Exploiting social contexts. Social contexts are widely exploited in recommender systems [161, 162, 163] and prediction models [164]. Both [162] and [163] added social regularization into matrix factorization recommendation model to constrain the taste difference while Lu [164] incorporated social contexts to a linear regression model for review quality predictions. However, it was not clear if social contexts can help data summarization and our work is the first work that incorporates social contexts into data summarization model for preserving topics in the original dataset.

Submodular Optimization. Submodularity [165] is a property of set functions which models natural diminishing returns property. This property states that adding an element to a smaller set has more value than adding it to a larger set. In data summarization, many previous studies [69, 70, 73, 134] designed their objective functions as submodular functions in order to achieve the performance guarantee (i.e., $1-1/e$ approximation guarantee to the optimum solution) for their greedy algorithms. In addition, many variants such as lazy greedy [146] and stochastic greedy [147] exploited submodularity to implement accelerated versions of classical greedy algorithm. In this work, we also design our objective function as a submodular function and apply the lazy greedy algorithm to accelerate the search performance.

6.6 Conclusion and Future Work

Summarizing social data gives us the opportunity to exploit social contexts for data summarization. In this chapter, we select Twitter as an example social data site, and focus on summarizing Twitter datasets while preserving topics. We firstly design a simple summarization model, namely *E-model*, that preserves topics without leveraging social contexts. Then, we propose our *S-model* which exploits two social contexts that are important for topic generation and dissemination. Finally, our experimental results demonstrate the effectiveness of our *S-model* for two different applications that are based on topic models.

This work suggests some interesting future directions. One direction is that we can extend our models by incorporating other social contexts. For example, we can explore friendships in the social network to help data summarization. Another direction is that we can explore other applications that are based on different analytical models other than topic model (which is what we focus on in this work). Also, it is an interesting question whether there exists a general model that can summarize data for different types of applications.

Part V

Conclusions

Chapter 7

Conclusion and Future Work

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill

7.1 Retrospective

Cloud computing has been the focus of IT decision makers for several years, but there are still many companies which are hesitant to move their data and workloads into the cloud. The single cloud model cannot meet all users' requirements on service latency, data privacy, location proximity, etc. Thus, many different types of multicloud are emerging and gaining a strength in business. In this thesis, we discuss four of them, namely geographically distributed cloud, federated cloud, hybrid cloud and decentralized cloud, with a focus on how these multicloud models complement to existing centralized cloud. Within the multicloud, we need to manage heterogeneous resources (*i.e.*, computing, storage and bandwidth) from different cloud providers in a cost-effective manner, which also gives us the opportunities to explore the multicloud resource allocation from the perspectives of cooperation, optimization and data sharing.

In the first part of this thesis, we explore how small data centers collaborate in a decentralized cloud. We propose a decentralized cloud model in which a group of networked SDCs can work collaboratively to overcome the limitations raised by massive centralized cloud infrastructure. We design different resource allocation strategies and evaluate their performance in a strategic setting. We discover that the reciprocity-based strategies are more effective than other strategies, which can help the SDCs improve the performance of cooperation.

The second part of this thesis focuses on optimization in a hybrid cloud using both private and public cloud resources. The objective of our multicloud optimization is to reduce information leakage to each cloud. In a multicloud storage system such as DepSky, distributing data on multiple clouds provides users with a certain degree of information leakage control in that no single cloud provider is privy to all the user's data. However, we demonstrate that unplanned distribution of data chunks can lead to avoidable information leakage. Consequently, we design StoreSim, an information leakage aware storage system, to optimize the information leakage in the multicloud environment. StoreSim achieves

this goal by using novel algorithms, BFSMinHash and SPClustering, which place the data with minimal information leakage (based on similarity) on the same cloud. Through an extensive evaluation based on two real datasets, we demonstrate that StoreSim is both effective and efficient (in terms of time and storage space) in minimizing information leakage during the process of synchronization in multicloud. Furthermore, through our attackability analysis, we show that StoreSim not only reduces the risk of wholesale information leakage but also makes attacks on retail information much more complex.

The last part of this thesis investigates the cost-effective data sharing among small data centers. On the one hand, we firstly design CoShare to schedule data transfers among small data centers with considering the tradeoff between cost and performance of massive data sharing. Through extensive experiments, we demonstrated that CoShare is effective in this mapping and guarantees the desirable cost and performance tradeoff for the data sharing process. On the other hand, we aim to reduce the total size of network transfer by making a small summary for big data. The smaller summary of a big dataset not only reduces the network transfer but also achieves savings in both bandwidth cost and transfer time. Specially, we select Twitter as an example social data site, and focus on summarizing Twitter datasets while preserving topics. We firstly design two simple summarization models that preserve topics without leveraging social contexts. Then, we propose our *S-model* which exploits two social contexts that are important for topic generation and dissemination. Finally, our experimental results demonstrate the effectiveness of our *S-model* for reducing the total size of data without the loss on application performance.

In summary, our work makes research efforts in (i) cooperation strategies for sharing compute resources in the decentralized cloud; (ii) reducing information leakage in the multicloud storage system; and (iii) scheduling network transfers in the data sharing network.

7.2 Perspective

Our work can be strengthened in many different directions, which gives many opportunities for future work. We highlight a few that we consider important to be addressed:

- **Blockchain-based decentralized cloud.** For the decentralized cloud, how will we motivate resource providers to participate? We can incentivize them to share idle resources simply by saving them time, cost and so on. In the centralized cloud computing, cloud providers develop billing systems to enable pay-as-you-go business models. Similarly, in the decentralized cloud, if there is also a billing system for resource providers that saves every transaction of resource usage history, they can be well motivated to share their resources in a pay-as-you-share manner. At present, blockchain technology paves the way for such financial billing systems using cryptographic building blocks, *e.g.*, BitCoin system. All transactions are saved in a public ledger and there is no centralized authority in this billing system. Furthermore, it is computationally infeasible for any single resource provider to modify history. With such decentralized billing systems, we can develop a good business model for the decentralized cloud. There are many interesting problems on how to reinvent the cloud based on blockchain technologies.

- **Multicloud privacy optimization.** In StoreSim, we measure information leakage based on syntactic similarity measures. One of the future work is to optimize data privacy in the multicloud

from the perspective of semantic measures. Protecting data privacy in the cloud is much more complex since privacy is a social notion with many facets. There are many definitions on the privacy that are proposed by the research community to understand the privacy. For example, the privacy can be defined either as the sensitive information such as identity or membership information or as the control over their sensitive information. Thus, to optimize the privacy in the multicloud environment, we can define user's data privacy based on the information they expect to protect in a semantic way. Furthermore, we can design distribution mechanisms to improve users' control over their sensitive data.

- **Data Transfer Management.** In CoShare, we optimize the transfer cost given the constraints on bandwidth and the transfer deadline. In the real world, we may have more constraints on the data transfer. For example, with increasing concerns on data privacy, many European countries are now concerned about their data leaving European borders and start to impose new regulatory constraints on data storage. Thus, we shall design our data sharing system to satisfy these new constraints. In addition, we can implement the CoShare system and provide a Data Transfer as a service. From the system perspective, we need to take more practical factors into account, such as network performance variation, low bandwidth utilization, scalability and fault tolerance of network failures. As a data sharing system in the multicloud, CoShare needs to solve the issues on the cloud interoperability, cross-platform transfer management, etc.

- **Making Big Data Smaller.** In our work, we employ data summarization approach to making big data smaller. For this work, we have two possible future directions. One direction is that we can extend our models by incorporating other social contexts while the other direction is that we can explore other applications that are based on different analytical models other than the topic model (which is what we focus on in our work). Besides data summarization approach, we can explore other models to reduce the total size of big data. For example, when the data is collected at the network edge, we can design a classifier or a scoring algorithm to evaluate the quality of data and to decide whether to abandon the data or not. Furthermore, this classifier or algorithm shall be capable of determining when and how to update themselves. For example, the topics in the tweets vary over the time and therefore the classifier needs to be updated as well so that they can better classify whether the current tweet is relevant or not. It is very interesting and challenging task to update these classifiers in a dynamic setting.

Bibliography

[1] J. Crowcroft, "On the duality of resilience and privacy," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 471, p. 20140862, The Royal Society, 2015. 4, 20, 57

[2] H. Harkous, R. Rahman, and K. Aberer, "C3p: Context-aware crowdsourced cloud privacy," in *14th Privacy Enhancing Technologies Symposium (PETS 2014)*, 2014. 4, 79

[3] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 12, 2013. 4, 22, 23, 24, 26, 29, 57, 60, 79

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50–58, 2010. 4, 16, 17, 35, 42

[5] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 1–14, ACM, 2010. 4, 16

[6] "Jcloud." `https://jclouds.apache.org/`. 5, 30

[7] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, 2009. 5, 29, 37

[8] "Symform cloud." `http://symform.com/`. 6

[9] "Storj cloud storage." `https://storj.io/`. 6, 30

[10] "Spot cloud." `http://www.virtustream.com/`. 6, 18

[11] "Dropbox." `https://www.dropbox.com/`. 17, 85

[12] S. Wilkinson, J. Lowry, and T. Boshevski, "Metadisk a blockchain-based decentralized file storage application," tech. rep., Technical Report. http://metadisk. org/metadisk. pdf, 2014. 20

[13] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," *IEEE Internet Computing*, vol. 17, no. 5, pp. 70–73, 2013. 20, 22, 36

[14] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015. 20

[15] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015. 20

[16] N. Samaan, "A novel economic sharing model in a federation of selfish cloud providers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013. 21, 22, 23, 25, 26, 36

[17] I. Petri, T. Beach, M. Zou, J. Diaz-Montes, O. Rana, and M. Parashar, "Exploring models and mechanisms for exchanging resources in a federated cloud," 21, 26, 36, 37

[18] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," *University of Toronto, Tech. Rep*, 2012. 21, 23, 37, 42

[19] A. N. Toosi, R. N. Calheiros, R. K. Thulasiram, and R. Buyya, "Resource provisioning policies to increase iaas provider's profit in a federated cloud environment," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 279–287, IEEE, 2011. 21, 26

[20] R. Pal and P. Hui, "Economic models for cloud service markets," in *Distributed Computing and Networking*, pp. 382–396, Springer, 2012. 22, 25

[21] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 229–240, ACM, 2010. 22

[22] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 292–308, ACM, 2013. 22, 23, 29, 57, 60, 79

[23] S. Di and C.-L. Wang, "Dynamic optimization of multiattribute resource allocation in self-organizing clouds," *IEEE Transactions on parallel and distributed systems*, vol. 24, no. 3, pp. 464–478, 2013. 22, 25

[24] H. Li, C. Wu, Z. Li, and F. Lau, "Profit-maximizing virtual machine trading in a federation of selfish clouds," *The University of Hong Kong, http://i.cs.hku.hk/ hxli/profit-federation. pdf, Tech. Rep*, 2012. 22

[25] H. Li, C. Wu, Z. Li, and F. C. Lau, "Profit-maximizing virtual machine trading in a federation of selfish clouds," in *INFOCOM, 2013 Proceedings IEEE*, pp. 25–29, IEEE, 2013. 22, 25

[26] M. Singhal, S. Chandrasekhar, T. Ge, R. S. Sandhu, R. Krishnan, G.-J. Ahn, and E. Bertino, "Collaboration in multicloud computing environments: Framework and security issues.," *IEEE Computer*, vol. 46, no. 2, pp. 76–84, 2013. 22

[27] M. Mihailescu and Y. M. Teo, "The impact of user rationality in federated clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 620–627, IEEE, 2012. 22

[28] O. Babaoglu, M. Marzolla, and M. Tamburini, "Design and implementation of a p2p cloud system," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 412–417, ACM, 2012. 22, 36

[29] T. Nguyen, Q. H. Vu, and R. Asal, "Harnessing the power of p2p technology to build pc2; an open and free cloud computing platform," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 646–651, 2012. 22

[30] H. Chen, Y. Hu, P. Lee, and Y. Tang, "Nccloud: A network-coding-based storage system in a cloud-of-clouds," 2013. 23, 29, 57, 60, 79

[31] T. G. Papaioannou, N. Bonvin, and K. Aberer, "Scalia: an adaptive scheme for efficient multi-cloud storage," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 20, IEEE Computer Society Press, 2012. 23, 28, 57, 60, 79

[32] "Emc hybrid cloud computing." `http://www.emc.com/cloud/hybrid-cloud-computing/index.htm`. 23, 60

[33] "Ibm multicloud toolkit." `http://www.zurich.ibm.com/csc/security/toolkit/`. 23, 60

[34] "Microsoft hybrid clouds." `http://www.microsoft.com/en-us/server-cloud/solutions/hybrid-cloud.aspx`. 23, 60

[35] S. Liu, S. Ren, G. Quan, M. Zhao, and S. Ren, "Profit aware load balancing for distributed cloud data centers," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 611–622, IEEE, 2013. 23, 25, 26

[36] Y. Feng, B. Li, and B. Li, "Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pp. 43–50, IEEE, 2012. 23, 27, 96

[37] E. Zhai, R. Chen, D. I. Wolinsky, and B. Ford, "Heading off correlated failures through independence-as-a-service," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 317–334, 2014. 23

[38] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "A hybrid edge-cloud architecture for reducing on-demand gaming latency," *Multimedia Systems*, pp. 1–17, 2014. 24, 79

[39] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Dynamic cost-efficient replication in data clouds," in *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, pp. 49–56, ACM, 2009. 24

[40] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Cost-efficient and differentiated data availability guarantees in data clouds," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 980–983, IEEE, 2010. 24

[41] H. Wang, F. Wang, J. Liu, and J. Groen, "Measurement and utilization of customer-provided resources for cloud computing," in *INFOCOM, 2012 Proceedings IEEE*, pp. 442–450, IEEE, 2012. 24, 25, 36

[42] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-datacenter transfers," in *Proceedings of the Tenth European Conference on Computer Systems*, p. 20, ACM, 2015. 24, 25, 26

[43] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. Lau, "Orchestrating bulk data transfers across geo-distributed datacenters," 24, 27, 96

[44] N. Yigitbasi, K. Datta, N. Jain, and T. Willke, "Energy efficient scheduling of mapreduce workloads on heterogeneous clusters," in *Green Computing Middleware on Proceedings of the 2nd International Workshop*, p. 1, ACM, 2011. 24, 26

[45] M. Zapater, J. L. Ayala, and J. M. Moya, "Leveraging heterogeneity for energy minimization in data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 752–757, IEEE, 2012. 24, 26

[46] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pp. 510–519, IEEE, 2013. 24

[47] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 323–336, 2015. 25

[48] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "Cyrus: Towards client-defined cloud storage," in *Proceedings of the Tenth European Conference on Computer Systems*, p. 17, ACM, 2015. 25

[49] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 22, IEEE Computer Society Press, 2012. 25

[50] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 228–235, IEEE, 2010. 25, 26

[51] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "Cyrus: Towards client-defined cloud storage," in *Proceedings of the Tenth European Conference on Computer Systems*, p. 17, ACM, 2015. 25

[52] V. Jalaparti and G. D. Nguyen, "Cloud resource allocation games," 2010. 25

[53] D. Niyato, A. V. Vasilakos, and Z. Kun, "Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 215–224, IEEE, 2011. 25

[54] P. Antoniadis, S. Fdida, T. Friedman, and V. Misra, "Federation of virtualized infrastructures: sharing the value of diversity," in *Proceedings of the 6th International COnference*, p. 12, ACM, 2010. 26

[55] M. Guazzone, C. Anglano, R. Aringhieri, and M. Sereno, "Distributed coalition formation in energy-aware cloud federations: A game-theoretic approach (extended version)," *CoRR, vol. abs/1309.2444*, 2013. 26

[56] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *INFOCOM, 2012 Proceedings IEEE*, pp. 702–710, IEEE, 2012. 26

[57] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Autonomic sla-driven provisioning for cloud applications," in *Proceedings of the 2011 11th IEEE/ACM international symposium on cluster, cloud and grid computing*, pp. 434–443, IEEE Computer Society, 2011. 26

[58] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram, "Delay tolerant bulk data transfers on the internet," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 229–238, ACM, 2009. 27, 83, 86, 96

[59] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 74–85, ACM, 2011. 27, 96

[60] Y. Wang, S. Su, A. X. Liu, and Z. Zhang, "Multiple bulk data transfers scheduling among datacenters," *Computer Networks*, vol. 68, 2014. 27, 96

[61] T. Kosar, E. Arslan, B. Ross, and B. Zhang, "Storkcloud: Data transfer scheduling and optimization as a service," in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pp. 29–36, ACM, 2013. 27, 84, 96

[62] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, and G. Antoniu, "Jetstream: Enabling high performance event streaming across cloud data-centers," in *Proceedings of the 8th ACM International Conference on Distributed Event-based Systems*, ACM, 2014. 27, 84, 96

[63] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul, "Efficient data transfer protocols for big data," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pp. 1–9, IEEE, 2012. 27, 96

[64] Z. Zhang, M. Zhang, A. G. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing cost and performance in online service provider networks.," in *NSDI*, pp. 33–48, 2010. 27, 96

[65] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, ACM, 2012. 27, 84, 96

[66] R. Tudoran, A. Costan, and G. Antoniu, "Transfer as a service: Towards a cost-effective model for multi-site cloud data management," in *SRDS*, 2014. 27, 96

[67] "The ranistor." `https://en.wikipedia.org/wiki/RainStor`. 28

[68] R. Zafarani, M. A. Abbasi, and H. Liu, *Social media mining: an introduction*. Cambridge University Press, 2014. 28

[69] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization: Identifying representative elements in massive data," in *Advances in Neural Information Processing Systems*, pp. 2049–2057, 2013. 28, 99, 100, 115, 116

[70] F. Pan, W. Wang, A. K. Tung, and J. Yang, "Finding representative set from massive data," in *Data Mining, Fifth IEEE International Conference on*, pp. 8–pp, IEEE, 2005. 28, 99, 100, 115, 116

[71] T. T. Nguyen, Q. V. H. Nguyen, M. Weidlich, and K. Aberer, "Result selection and summarization for web table search," in *31st IEEE International Conference on Data Engineering*, no. EPFL-CONF-203577, 2015. 28, 99, 100, 115

[72] R. Mehrotra and E. Yilmaz, "Representative & informative query selection for learning to rank using submodular functions," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 545–554, ACM, 2015. 28, 99, 115

[73] H. Lin and J. Bilmes, "A class of submodular functions for document summarization," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 510–520, Association for Computational Linguistics, 2011. 28, 99, 115, 116

[74] "Aspera." `http://asperasoft.com/`. 28

[75] "Cloud opt." `http://www.cloudopt.com/`. 28

[76] "Cloud beam." `http://www.attunity.com/products/cloudbeam`. 28

[77] "Cloud beam performance." `http://www.attunity.com/performance-details`. 28

[78] "The cloudspaces project." `http://www.cloudspaces.eu`. 30

[79] "The supercloud project." `https://supercloud-project.eu/`. 30

[80] P. De Filippi and S. McCARTHY, "Cloud computing: Legal issues in centralized architectures," in *VII International Conference on Internet, Law and Politics*, 2011. 35

[81] "Prism surveillance program by nsa." `http://en.wikipedia.org/wiki/PRISM_(surveillance_program)`. 35

[82] "Business cluster." http://en.wikipedia.org/wiki/Business_cluster. 35

[83] M. Van Alstyne, E. Brynjolfsson, and S. Madnick, "Why not one big database? principles for data ownership," *Decision Support Systems*, vol. 15, no. 4, pp. 267–284, 1995. 36

[84] T. Locher, R. Meier, R. Wattenhofer, and S. Schmid, "Robust live media streaming in swarms," in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, pp. 121–126, ACM, 2009. 37, 43

[85] R. M. Axelrod, *The evolution of cooperation*. Basic books, 2006. 37, 43

[86] K. Wang, M. Lin, F. Ciucua, A. Wierman, and C. Lin, "Characterizing the impact of the workload on the value of dynamic resizing in data centers," in *INFOCOM, 2013 Proceedings IEEE*, pp. 515–519. 38, 39

[87] H. Li, M. Muskulus, and L. Wolters, "Modeling job arrivals in a data-intensive grid," in *Job Scheduling Strategies for Parallel Processing*, pp. 210–231, Springer, 2007. 38, 39

[88] S. Mirtchev and R. Goleva, "Evaluation of pareto/d/1/k queue by simulation," 2008. 39

[89] R. Rahman, T. Vinkó, D. Hales, J. Pouwelse, and H. Sips, "Design space analysis for modeling incentives in distributed systems," in *ACM SIGCOMM 2011*, ACM, 2011. 39

[90] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *ACM SIGecom Exchanges*, vol. 5, pp. 41–50, 2005. 40, 41

[91] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A secure economic framework for peer-to-peer resource sharing," in *Workshop on Economics of Peer-to-Peer Systems*, vol. 35, 2003. 42

[92] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th international conference on World Wide Web*, pp. 640–651, ACM, 2003. 42

[93] A. Rapoport, *Prisoner's dilemma: A study in conflict and cooperation*, vol. 165. University of Michigan Press, 1965. 43

[94] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "Bass: Bittorrent assisted streaming system for video-on-demand," in *Multimedia Signal Processing, 2005 IEEE 7th Workshop on*, pp. 1–4, IEEE, 2005. 43

[95] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, pp. 23–50, 2011. 43

[96] G. Greenwald and E. MacAskill, "Nsa prism program taps in to user data of apple, google and others," *The Guardian*, vol. 7, no. 6, pp. 1–43, 2013. 57, 60

[97] T. Suel and N. Memon, "Algorithms for delta compression and remote file synchronization," 2002. 57, 61, 69

[98] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking personal cloud storage," in *Proceedings of the 2013 conference on Internet measurement conference*, pp. 205–212, ACM, 2013. 57, 58, 61, 79

[99] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, pp. 481–494, ACM, 2012. 58, 79

[100] U. Manber *et al.*, "Finding similar files in a large file system.," in *Usenix Winter*, vol. 94, pp. 1–10, 1994. 58

[101] "Prism surveillance program by nsa." `http://en.wikipedia.org/wiki/Edward_Snowden#Disclosure`. 60

[102] J. W. Hunt and M. MacIlroy, *An algorithm for differential file comparison*. Bell Laboratories, 1976. 60

[103] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, ACM, 2002. 64

[104] M. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 284–291, ACM, 2006. 64, 79

[105] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011. 64

[106] P. Li and C. König, "b-bit minwise hashing," in *Proceedings of the 19th international conference on World wide web*, pp. 671–680, ACM, 2010. 65, 79

[107] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1157–1166, 1997. 65

[108] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004. 66

[109] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping multidimensional data*, pp. 25–71, Springer, 2006. 67

[110] C.-Y. Chan and Y. E. Ioannidis, "Bitmap index design and evaluation," in *ACM SIGMOD Record*, vol. 27, pp. 355–366, ACM, 1998. 68

[111] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web*, pp. 141–150, ACM, 2007. 69, 72, 79

[112] "Murmur hash function." `https://sites.google.com/site/murmurhash/`. 70

[113] "Spideroak encryption specification." `https://spideroak.com/engineering_matters#encryption`. 78

[114] S. Harispe, S. Ranwez, S. Janaqi, and J. Montmain, "Semantic similarity from natural language and ontology analysis," *Synthesis Lectures on Human Language Technologies*, vol. 8, no. 1, pp. 1–254, 2015. 78

[115] C. Gentry, *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. 78

[116] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud storage with minimal trust," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 4, p. 12, 2011. 79

[117] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "Sporc: Group collaboration using untrusted cloud resources.," in *OSDI*, vol. 10, pp. 337–350, 2010. 79

[118] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *ACM SIGOPS Operating Systems Review*, vol. 35, pp. 202–215, ACM, 2001. 79

[119] L. P. Cox and B. D. Noble, "Samsara: Honor among thieves in peer-to-peer storage," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 120–132, 2003. 79

[120] T. Zou, R. Le Bras, M. V. Salles, A. Demers, and J. Gehrke, "Cloudia: a deployment advisor for public clouds," in *Proceedings of the VLDB Endowment*, vol. 6, pp. 121–132, VLDB Endowment, 2012. 79

[121] J.-M. Bohli, N. Gruschka, M. Jensen, L. L. Iacono, and N. Marnau, "Security and privacy-enhancing multicloud architectures," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 4, pp. 212–224, 2013. 79

[122] I. Ion, N. Sachdeva, P. Kumaraguru, and S. Čapkun, "Home is safer than the cloud!: privacy concerns for consumer cloud storage," in *Proceedings of the Seventh Symposium on Usable Privacy and Security*, p. 13, ACM, 2011. 79

[123] T. Li and N. Li, "On the tradeoff between privacy and utility in data publishing," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 517–526, ACM, 2009. 79

[124] A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE multimedia*, no. 4, pp. 8–13, 2007. 83

[125] R. Peterson and E. G. Sirer, "Antfarm: Efficient content distribution with managed swarms.," in *NSDI*, vol. 9, pp. 107–122, 2009. 84, 85, 96

[126] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, pp. 68–72, 2003. 84

[127] H. Zhuang, R. Rahman, and K. Aberer, "Decentralizing the cloud: How can small data centers cooperate?," in *Peer-to-Peer Computing (P2P), 14-th IEEE International Conference on*, pp. 1–10, Ieee, 2014. 85

[128] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008. 86

[129] "Mosek lp solver." `https://www.mosek.com/`. 91

[130] L.-H. Vu, M. Hauswirth, and K. Aberer, "Qos-based service selection and ranking with trust and reputation management," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pp. 466–483, Springer, 2005. 95

[131] M. Capota, N. Andrade, T. Vinkó, F. Santos, J. Pouwelse, and D. Epema, "Inter-swarm resource allocation in bittorrent communities," in *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pp. 300–309, IEEE, 2011. 96

[132] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pp. 66–66, IEEE, 2006. 96

[133] H. Zhuang, I. Filali, R. Rahman, and K. Aberer, "Coshare: A cost-effective data sharing system for data center networks," in *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, pp. 11–18, IEEE, 2015. 99

[134] H. P. Vanchinathan, A. Marfurt, C.-A. Robelin, D. Kossmann, and A. Krause, "Discovering valuable items from massive data," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1195–1204, ACM, 2015. 99, 116

[135] X. Zhu, C. Vondrick, D. Ramanan, and C. Fowlkes, "Do we need more training data or better models for object detection?.," in *BMVC*, vol. 3, p. 5, Citeseer, 2012. 99

[136] S.-A. Bahrainian and A. Dengel, "Sentiment analysis and summarization of twitter data," in *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pp. 227–234, IEEE, 2013. 100, 115

[137] D. Chakrabarti and K. Punera, "Event summarization using tweets.," *ICWSM*, vol. 11, pp. 66–73, 2011. 100, 115

[138] J. Nichols, J. Mahmud, and C. Drews, "Summarizing sporting events using twitter," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pp. 189–198, ACM, 2012. 100, 115

[139] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003. 100, 101, 102, 103, 111, 115

[140] Q. Mei, D. Cai, D. Zhang, and C. Zhai, "Topic modeling with network regularization," in *Proceedings of the 17th international conference on World Wide Web*, pp. 101–110, ACM, 2008. 101

[141] Y. Cha, B. Bi, C.-C. Hsieh, and J. Cho, "Incorporating popularity in topic models for social network analysis," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pp. 223–232, ACM, 2013. 101

[142] S. Fujishige, "Polymatroidal dependence structure of a set of random variables," *Information and Control*, vol. 39, no. 1, pp. 55–72, 1978. 104

[143] Y. Chang, X. Wang, Q. Mei, and Y. Liu, "Towards twitter context summarization with user influence models," in *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 527–536, ACM, 2013. 105, 115, 116

[144] J. Surowiecki, *The wisdom of crowds*. Anchor, 2005. 106

[145] S. Auty and R. Elliott, "Being like or being liked: identity vs. approval in a social context.," *Advances in Consumer Research*, vol. 28, no. 1, 2001. 107

[146] M. Minoux, "Accelerated greedy algorithms for maximizing submodular set functions," in *Optimization Techniques*, pp. 234–243, Springer, 1978. 108, 116

[147] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause, "Lazier than lazy greedy," *arXiv preprint arXiv:1409.7938*, 2014. 108, 116

[148] "Twitter public apis." `https://dev.twitter.com/overview/documentation`. 109

[149] "Twitter public search apis." `https://dev.twitter.com/rest/public/search`. 109

[150] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: Massive data summarization on the fly," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 671–680, ACM, 2014. 109

[151] A. Haghighi and L. Vanderwende, "Exploring content models for multi-document summarization," in *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 362–370, Association for Computational Linguistics, 2009. 109

[152] J. Steinberger and K. Jezek, "Using latent semantic analysis in text summarization and summary evaluation," in *Proc. ISIM'04*, pp. 93–100, 2004. 109

[153] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, pp. 457–479, 2004. 109

[154] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis, "Discovering geographical topics in the twitter stream," in *Proceedings of the 21st international conference on World Wide Web*, pp. 769–778, ACM, 2012. 111, 115

[155] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Who, where, when and what: discover spatio-temporal topics for twitter users," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 605–613, ACM, 2013. 111, 115

[156] W. X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, and X. Li, "Comparing twitter and traditional media using topic models," in *Advances in Information Retrieval*, pp. 338–349, Springer, 2011. 111, 115

[157] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley, "Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose," *arXiv preprint arXiv:1306.5204*, 2013. 111

[158] "Machine learning for language toolkit." `http://mallet.cs.umass.edu/`. 111

[159] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys (CSUR)*, vol. 18, no. 1, pp. 23–38, 1986. 112

[160] L. Hong and B. D. Davison, "Empirical study of topic modeling in twitter," in *Proceedings of the First Workshop on Social Media Analytics*, pp. 80–88, ACM, 2010. 115

[161] X. Hu, L. Tang, J. Tang, and H. Liu, "Exploiting social relations for sentiment analysis in microblogging," in *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 537–546, ACM, 2013. 116

[162] X. Liu and K. Aberer, "Soco: a social network aided context-aware recommender system," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 781–802, International World Wide Web Conferences Steering Committee, 2013. 116

[163] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 287–296, ACM, 2011. 116

[164] Y. Lu, P. Tsaparas, A. Ntoulas, and L. Polanyi, "Exploiting social context for review quality prediction," in *Proceedings of the 19th international conference on World wide web*, pp. 691–700, ACM, 2010. 116

[165] B. Sankaran, M. Ghazvininejad, X. He, D. Kale, and L. Cohen, "Learning and optimization with submodular functions," *arXiv preprint arXiv:1505.01576*, 2015. 116

Hao Zhuang

Education

2012– **PhD candidate**, *Distributed Information Systems Lab*, EPFL, Switzerland.

2009–2012 **Master of Engineering**, *Service Science and Engineering*, Peking University, China.

2011–2012 **MSc.**, *Communication System Design*, Royal Institute of Technology, Sweden.

2010–2012 **MSc.**, *Information Security and Network Services*, Aalto University, Finland .

2005–2009 **Bachelor of Engineering**, *Software Engineering*, Northeastern University, China.

Research

2013-present **Big data analytics in the Cloud**, *Cloud platform provides a large computation and storage infrastructure to ensure successful data processing and analysis. In this research, I focus on two types of data analytic tasks.*

- Prediction task based on time series data about resource usage in Google cluster. We analyze the resource usage of Google cluster such as CPU, memory and disk and develop resource usage prediction models based on (linear and non-linear) regression models and artificial neural network model;
- Data summarization task for social media data from Twitter. We develop a data summarization framework to summarize tweets with a smaller subset which can preserve topics in the original big dataset.

2013-present **Decentralized Cloud**, *Most cloud providers are centralized entities that employ massive data centers. However, in recent times, due to increasing concerns about privacy and data control, many small data centers (SDCs) established by different providers are emerging in an attempt to avoid severe issues (e.g., legal, privacy, and data control) that can arise with the adoption of massive centralized data centers. In this research, I focus on three dimensions.*

- Client-defined multicloud storage systems in which we design efficient data placement algorithms to optimize different dimensions in the multicloud, e.g., reducing information leakage, improving data availability and overcoming vendor lock-in;
- Efficient resource allocation in a collaborative multicloud environment which we evaluate different cooperation strategies and find out the effective strategies for small data centers to cooperate;
- Cost-effective data sharing among small data centers in which we aim to manage the tradeoff between the cost and performance of data sharing and provide guarantees on the data sharing performance.

Avenue du Tir-Fédéral 25 – Ecublens 1024
(0041) 789 466 886 • ☎ (0041) 216 935 257
hao.zhuang@epfl.ch

2012-2014 **Impact of Virtualization on Performance of Cloud Data Centers**, *Public cloud platforms might start with homogeneous hardware; nevertheless, because of inevitable hardware upgrades, or adding more capacity, the initial homogeneous platform will gradually evolve into heterogeneous as time passes by. The consequent performance heterogeneity is of concern to cloud users. In this research, we evaluate performance variations from hardware heterogeneity and scheduling mechanisms of public clouds e.g. Amazon EC2 and Rackspace. A comprehensive set of microbenchmarks and application-level macrobenchmarks have been used to investigate performance variation. Based on observations, we propose several strategies to help users find better-performing instances*.

Publications

1 **H. Zhuang**, R. Rahman, X. Hu, T. Guo, H. Pan and K. Aberer. Data Summarization with Social Contexts. 25th ACM International Conference on Information and Knowledge Management (CIKM 2016)

2 **H. Zhuang**, R. Rahman, H. Pan and K. Aberer. Optimizing Information Leakage in Multicloud Storage Services. in IEEE Transactions on Cloud Computing, 2016.

3 **H. Zhuang**, R. Rahman, H. Pan and K. Aberer. StoreSim: Optimizing Information Leakage in Multicloud Storage Services. 7th IEEE International Conference on Cloud Computing Technology and Science, Vancouver, BC, Canada, 2015. (**Best Student Paper Award**)

4 T. Guo, J.-P. Calbimonte, **H. Zhuang** and K. Aberer. SigCO: Mining Significant Correlations via a Distributed Real-time Computation Engine. 2015 IEEE International Conference on Big Data, SANTA CLARA, 2015.

5 **H. Zhuang**, I. Filali, R. Rahman and K. Aberer. CoShare: A Cost-effective Data Sharing System for Data Center Networks. 2015 IEEE International Conference on Collaboration and Internet Computing, Hangzhou, China, 2015.

6 **H. Zhuang**, R. Rahman and K. Aberer. Decentralizing the Cloud: How Can Small Data Centers Cooperate? 14th IEEE International Conference on Peer-to-Peer Computing (P2P), 2014.

7 T. Guo, T. G. Papaioannou, **H. Zhuang** and K. Aberer. Online indexing and distributed querying model-view sensor data in the cloud. The 19th International Conference on Database Systems for Advanced Applications (DASFAA), 2014.

8 Z. Ou, **H. Zhuang**, A. Lukyanenko, J. K. Nurminen and P. Hui et al. Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds, in IEEE Transactions on Cloud Computing, 2013.

9 **H. Zhuang**, X. Liu, Z. Ou and K. Aberer. Impact of Instance Seeking Strategies on Resource Allocation in Cloud Data Centers. Sixth International Conference on Cloud Computing (CLOUD '13), San Jose, USA, 2013.

10 Z. Ou, **H. Zhuang**, J.K. Nurminen, A. Yla-Jaaski, P. Hui. Exploiting hardware heterogeneity within the same instance type of Amazon EC2. 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12). [**Citation 62**, covered by BBC News Technology, The Register, ACM TechNews, etc.]

11 Z. Ou, **H. Zhuang**, A. Yla-Jaaski. How much energy can you save? an energy perspective of youtube. Poster in HotCloud 12, 1-2.

12 **H. Zhuang**, H. Ntareme, Z. Ou, B. Pehrson. A service adaptation middleware for delay tolerant networks based on http simple queue service. Proceedings of the 6th USENIX Workshop on Networked Systems for Developing Regions (NSDR 12).

13 **H. Zhuang**, Xin Gu. Security Analysis on Linux Container as a Lightweight virtualization solution for Cloud Computing, Student Poster presentation at the 16th Nordic Conference on Secure IT Systems, 2011

Project Experience

02/2013-05/2013 **Recommendation System for Netflix Challenges**,
Hadoop implementation of recommendation algorithm based on Matrix factorization.

09/2011-01/2012 **Delay Tolerant Network (DTN) based on Android**,
Open source DTN implementation on Android phone.

02/2011-06/2011 **Event Finder Project based on Android**,
Web services mashup for Facebook Event API and Helsinki Transport API .

09/2009-07/2010 **Context-Aware Services for indoor location**,
Indoor location modeling and navigation for Copenhagen airport, implemented by Java .

09/2009-12/2009 **Microblog service: followme website**,
Develop microblog website based on J2EE framework (Struts2/Spring/Hibernate) with JQuery and HTML in the front-end .

09/2008-06/2009 **Search Engine for Patent Information**,
Patent search engine based on Nutch, implemented by ASP.NET framework.

Awards

2016 Travel grant in ACM CIKM 2016

2015 Best Student Paper Award at IEEE CloudCom 2015

2012 Graduate with distinction for both master degrees

2010-2012 Erasmus Mundus full scholarship by European Commission (48,000 EUR)

2009 Admitted by Peking University with National Entrance Examination waived

2009 Outstanding Graduate of Northeastern University (top 3%)

2005-2009 First class scholarship of Northeastern University for 2 times (top 3%), second for 2 times (top 10%)

Professional Services

External Reviewer: IEEE BigData Congress 2016, IEEE BigData Congress 2015, WISE 2016, WISE 2015, ICDE 2015, ICDE 2014, IEEE P2P 2015, ICDCS 2013

Languages

Chinese **Mother-tongue**

English **Proficient**

References

Karl Aberer Professor and Vice President in EPFL, karl.aberer@epfl.ch

Rameez Rahman Research Scientist in Bell Labs, rameez.rahman@nokia.com

Weiping Li Professor in Peking University, wpli@ss.pku.edu.cn

Pan Hui Professor in Hong Kong University of Science and Technology, panhui@cse.ust.hk

Avenue du Tir-Fédéral 25 – Ecublens 1024
(0041) 789 466 886 • ☎ (0041) 216 935 257
✉ hao.zhuang@epfl.ch